Разработчик

Обзор

Обзор

Управление пространствами имён

Управление жизненным циклом приложений

Управление рабочими нагрузками Kubernetes

Быстрый старт

Creating a simple application via image

Introduction

Important Notes

Prerequisites

Workflow Overview

Procedure

Создание приложений

Построение архитектуры при

Введение в построение приложения

Основные компоненты

Основные понятия

Создание приложений

Пространст

Обзор концепц

Эксплуатация и сопровожден

Рабочие на

Работа с Helm charts

1. Понимание Helm

2 Развертывание Helm Charts как прило

3. Развертывание Helm Charts как прил

Наблюдаемость приложений

Конфигура

Как сделат

Образы

Обзор образов

Понимание контейнеров и образов

Образы

Реестр образов

Репозиторий образов

Теги образов

Идентификаторы образов

Контейнеры

Как сделать

Реестр

Введение

Принципы и изоляция по namespace

Аутентификация и авторизация

Преимущества

Сценарии применения

Установка

Обновление

Как сделат

Source to Image

Обзор

Установка

Обновлени

Руководства

Как сделать

Стратегия изоляции узлов

Введение

Преимущества

Сценарии применения

Архитектура

Руководства

Основные

Разрешени

Часто задаваемые вопросы

Часто задаваемые вопросы

Почему не должно быть нескольких ResourceQuota в одном namespace при его импорте?

Почему не должно быть нескольких LimitRange в одном namespace при его импорте?

Обзор

Alauda Container Platform предоставляет единый интерфейс для создания, редактирования, удаления и управления облачными нативными приложениями как через веб-консоль, так и через CLI (Command-Line Interface). Приложения могут быть развернуты в нескольких пространствах имён с применением политик RBAC.

Содержание

Управление пространствами имён

Пространства имён обеспечивают логическую изоляцию ресурсов Kubernetes. Основные операции включают:

- Создание пространств имён: Определение квот ресурсов и политик допуска безопасности Pod.
- Импорт пространств имён: Импорт существующих пространств имён Kubernetes в Alauda Container Platform обеспечивает полное соответствие функциональности

платформы с нативно созданными пространствами имён.

Управление жизненным циклом приложений

Alauda Container Platform поддерживает сквозное управление жизненным циклом, включая:

Шаблоны создания приложений

В Alauda Container Platform приложения можно создавать различными способами. Вот некоторые распространённые методы:

- Создание из образов: Создание пользовательских приложений с использованием предварительно собранных контейнерных образов. Этот метод поддерживает создание полного приложения, включающего `Deployments`, `Services`, `ConfigMaps` и другие ресурсы Kubernetes.
- Создание из каталога: Alauda Container Platform предоставляет каталоги приложений, позволяя пользователям выбирать предопределённые шаблоны приложений (Helm Charts или Operator Backed) для создания.
- Создание из YAML: Импортируя YAML-файл, создайте пользовательское приложение со всеми включёнными ресурсами за один шаг.
- Создание из кода: Сборка образов с помощью Source to Image (S2I).

Операции с приложениями

- Обновление приложений: Обновление версии образа приложения, переменных окружения и других конфигураций, либо импорт существующих ресурсов Kubernetes для централизованного управления.
- Экспорт приложений: Экспорт приложений в форматах YAML, Kustomize или Helm Chart, с последующим импортом для создания новых экземпляров приложений в других пространствах имён или кластерах.
- Управление версиями: Поддержка автоматического или ручного создания версий приложений, а в случае проблем доступен однокликовый откат к конкретной версии для быстрого восстановления.

- Удаление приложений: Удаление приложения одновременно удаляет само приложение и все его напрямую включённые ресурсы Kubernetes. Кроме того, эта операция разрывает любые связи приложения с другими ресурсами Kubernetes, которые не были напрямую частью его определения.

Наблюдаемость приложений

Для непрерывного управления работой платформа предоставляет логи, события, мониторинг и др.

- Логи: Поддержка просмотра логов в реальном времени из текущего запущенного Pod, а также логов предыдущих перезапусков контейнеров.
- События: Поддержка просмотра информации о событиях для всех ресурсов в пространстве имён.
- Мониторинговые панели: Предоставление мониторинговых панелей на уровне пространства имён, включая специализированные представления для приложений, Workloads и Pods, а также поддержку настройки панелей мониторинга под конкретные операционные требования.

Управление рабочими нагрузками Kubernetes

Поддержка основных типов рабочих нагрузок:

- Deployments: Управление безсостояниями приложениями с возможностью rolling updates.
- StatefulSets: Запуск stateful-приложений со стабильными сетевыми идентификаторами.
- DaemonSets: Развёртывание сервисов на уровне узлов (например, сборщиков логов).
- CronJobs: Планирование пакетных заданий с политиками повторных попыток.

Быстрый старт

Creating a simple application via image

Creating a simple application via image

> В этом техническом руководстве показано, как эффективно создавать, управлять и получать доступ к контейнеризованным приложениям в Alauda Container Platform с использованием нативных для Kubernetes методов.

Содержание

Introduction

Use Cases

- Новые пользователи, желающие понять основные рабочие процессы создания приложений на платформах Kubernetes
- Практическое упражнение, демонстрирующее основные возможности платформы, включая:
  - Организацию проектов/пространств имён
  - Создание Deployment
  - Шаблоны экспонирования сервисов
  - Проверку доступности приложения

Time Commitment

Оценочное время выполнения: 10-15 минут

Important Notes

- Это техническое руководство сосредоточено на основных параметрах — для расширенных настроек обращайтесь к полной документации
- Необходимые разрешения:
  - Создание проектов/пространств имён
  - Интеграция репозитория образов
  - Развёртывание workloads

Prerequisites

- Базовое понимание архитектуры Kubernetes и концепций платформы Alauda Container Platform
- Предварительно настроенный проект согласно процедурам создания платформы

Workflow Overview

No.	Operation	Description
1	Create Namespace	Создание границы изоляции ресурсов
2	Configure Image Repository	Настройка источников контейнерных образов
3	Create application via Deployment	Создание workload Deployment
4	Expose Service via NodePort	Настройка сервиса типа NodePort
5	Validate Application Accessibility	Проверка доступности приложения

Procedure

Create namespace

> Пространства имён обеспечивают логическую изоляцию для группировки ресурсов и управления квотами.

Prerequisites

- Разрешения на создание, обновление и удаление пространств имён (например, роли Administrator или Project Administrator)
- kubectl, настроенный с доступом к кластеру

Creation Process

1. Войдите в систему и перейдите в **Project Management** > **Namespaces**
2. Выберите **Create Namespace**
3. Настройте основные параметры:

** Parameter **	Описание
Cluster	Целевой кластер из связанных с проектом кластеров
Namespace	Уникальный идентификатор (автоматически с префиксом имени проекта)

4. Завершите создание с настройками ресурсов по умолчанию

Configure Image Repository

Alauda Container Platform поддерживает несколько стратегий получения образов:

Method 1: Integrated Registry via Toolchain

1. Перейдите в **Administrator** > **Toolchain** > **Integration**
2. Создайте новую интеграцию:

Parameter	Требование
Name	Уникальный идентификатор интеграции
API Endpoint	URL сервиса реестра (HTTP/HTTPS)
Secret	Существующие или новые учётные данные

3. Назначьте реестр целевому проекту платформы

Method 2: External Registry Services

- Используйте общедоступные URL реестров
- Пример: `nginx:latest`

Registry Access and Authentication

Для успешного скачивания контейнерных образов кластером необходимо выполнить два ключевых условия:

- **Сетевой доступ**: Узлы кластера должны иметь исходящий (egress) сетевой доступ к конечным точкам реестра контейнеров. При отсутствии соединения могут возникать ошибки `ImagePullBackOff` или другие сетевые ошибки.
- **Аутентификация**: Если реестр требует аутентификации, создайте `ImagePullSecret` и укажите его в Pod или Deployment, чтобы кластер мог скачивать образы. Подробные инструкции см. в документации по Image Pull Secrets.

Create application via Deployment

> Deployment обеспечивает декларативное обновление реплик Pod.

Creation Process

1. В представлении **Container Platform**:
   - Используйте селектор namespace для выбора целевой границы изоляции
2. Перейдите в **Workloads** > **Deployments**
3. Нажмите **Create Deployment**
4. Укажите источник образа:
   - Выберите интегрированный реестр *или*
   - Введите внешний URL образа (например, `nginx:latest`)
5. Настройте идентификацию workload и запустите

Management Operations

- Мониторинг статуса реплик
- Просмотр событий и логов
- Просмотр YAML-манифестов
- Анализ метрик ресурсов, оповещений

Expose Service via NodePort

Сервисы обеспечивают сетевой доступ к группам Pod.

Creation Process

1. Перейдите в **Networking** > **Services**
2. Нажмите **Create Service** с параметрами:

Parameter	Значение
Type	NodePort
Selector	Имя целевого Deployment
Port Mapping	Порт сервиса: Порт контейнера (например, 8080:80)

3. Подтвердите создание.

Critical

- Виртуальный IP, видимый в кластере
- Диапазон выделения NodePort (30000-32767)

Внутренние маршруты обеспечивают обнаружение сервисов для workloads, предоставляя единый IP-адрес или порт хоста для доступа.

1. Перейдите в **Network** > **Service**.
2. Нажмите **Create Service**.
3. Настройте **Details** согласно параметрам ниже, остальные параметры оставьте по умолчанию.

Parameter	Описание
Name	Введите имя сервиса.
Type	`NodePort`

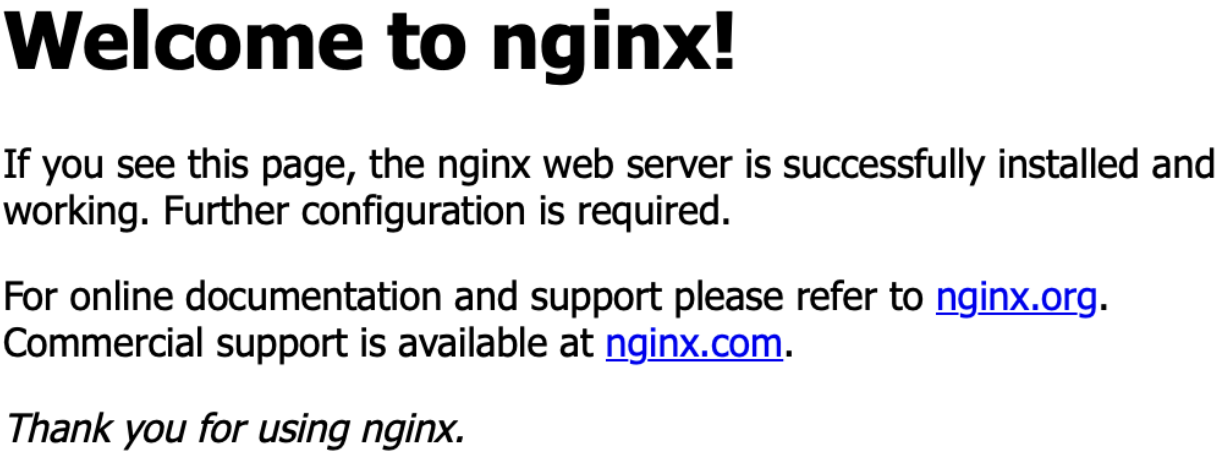
Parameter	Описание
Workload Name	Выберите ранее созданный `Deployment`.
Port	**Service Port**: номер порта, который сервис открывает внутри кластера, то есть Port, например, `8080`. **Container Port**: целевой номер порта (или имя), сопоставленный с портом сервиса, то есть targetPort, например, `80`.

4. Нажмите **Create**. На этом этапе сервис успешно создан.

Validate Application Accessibility

Verification Method

1. Получите компоненты открытого эндпоинта:
   - **Node IP**: публичный адрес рабочего узла
   - **NodePort**: выделенный внешний порт
2. Сформируйте URL доступа: `http://<Node_IP>:<NodePort>`
3. Ожидаемый результат: страница приветствия Nginx



Создание приложений

Построение архитектуры приложения

Построение архитектуры приложения

Введение в построение приложения

Основные компоненты

Основные понятия

Типы приложений

Custom Applications

Понимание пользовательских приложении

Типы нагру

Понимание параметров

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использов

Рекомендации по лучшим практикам

Устранение распространённых проблем

Расширенные сценарии использования

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Понимание

Overview

Core Concepts

Примеры CLI и практическое использова

Лучшие практики

Расширенные шаблоны использования

Описание е

Пространства имён

Создание пространств имён

Понимание пространств имён

Создание пространств имён через веб-к

Создание пространства имён в проекте

Создание пространства имён через **kub**

Импорт пространств имён

Overview

Use Cases

Prerequisites

Procedure

Resource Q

Понимание Re

Квоты

Квоты ресурсо

Политики безопасности Pod

Назначение UID/GID

Включение назначения UID/GID

Проверка назначения UID/GID

Управление участниками пространства имё

Обновление Namespaces

Обновление Quotas

Обновление Container LimitRanges

Обновление Pod Security Admission

Удаление/исключение пространств имён

Создание приложений

Эксплуатация и сопровождение приложений

Приложения

Понимание Ho

Предварительны

Запуск и остановка приложен

Запуск приложения

Остановка приложения

Настройка VerticalPodAutoscaler (VPA)

Обзор

Понимание VerticalPodAutoscalers

Предварительные требования

Создание VerticalPodAutoscaler

ка

Cr

елы

Cron

Объяснение пр

Обновление приложений

Импорт ресурсов

Удаление/пакетное удаление ресурсов

Экспорт приложений

Обновление и удаление Chart-приложений

Важные замечания

Предварительные требования

Описание анализа статуса

Управлени

Создание сним

Откат и истори

Удаление приложений

Обработка ошибок нехватки ресурсов

Overview

Настройка политик эвакуации

Создание политик эвакуации в конфигурации узла

Сигналы эвакуации

Пороговые значения эвакуации

Настройка доступных ресурсов для планирования

Предотвращение колебаний состояния узла

Освобождение ресурсов на уровне узла

Эвакуация подов

Качество обслуживания и Out of Memory Killer

Планировщик и условия нехватки ресурсов

Проверки с

Понимание пр

Пример YAML-

Параметры на

Устранение не

Пример сценария

Рекомендуемые практики

Рабочие нагрузки

Deployments

Understanding Deployments

Creating Deployments

Managing Deployments

Troubleshooting by using CLI

DaemonSets

Понимание DaemonSets

Создание DaemonSets

Управление DaemonSets

StatefulSets

Понимание Sta

Создание State

Управление St

Pods

Понимание Pod'ов

Пример YAML файла

Управление Pod с помощью CLI

Управление Pod через веб-консоль

Контейнер

Понимание кон

Понимание эф

Взаимодействи

Работа с Helm charts

Работа с Helm charts

1. Понимание Helm

2 Развертывание Helm Charts как приложений через CLI

3. Развертывание Helm Charts как приложений через UI

Конфигурации

Настройка ConfigMap

Понимание ConfigMap

Ограничения ConfigMap

Пример ConfigMap

Создание ConfigMap через веб-консоль

Создание ConfigMap с помощью CLI

Операции

Просмотр, редактирование и удаление

Способы использования ConfigMap в Pc

ConfigMap и Secret

Настройка Secrets

Понимание Secrets

Создание Secret типа Opaque

Создание Secret для реестра контейнеров

Создание Secret типа Basic Auth

Создание Secret типа SSH-Auth

Создание Secret типа TLS

Создание Secret через веб-консоль

Как использовать Secret в Pod

Последующие действия

Операции

Наблюдаемость приложений

Мониторинговые панели

Предварительные требования

Панели мониторинга на уровне Namespa

Мониторинг на уровне нагрузки

Логи

Процедура

События

Процедура

Интерпретация

Как сделать

Настройка правил срабатыва

Преобразование времени

Запись выражений Crontab

Добавление ImagePullSecrets в ServiceAcco

Создание ImagePullSecret

Добавление ImagePullSecret в ServiceAccount

Проверка установки imagePullSecrets для новых Pod

Построение архитектуры приложения

Содержание

Введение в построение приложения

Alauda Container Platform — это платформа для разработки и запуска контейнеризованных приложений. Она разработана для того, чтобы приложения и поддерживающие их дата-центры могли масштабироваться от нескольких машин и приложений до тысяч машин, обслуживающих миллионы клиентов.

Построенная на Kubernetes, Alauda Container Platform использует ту же надежную технологию, которая лежит в основе крупных телекоммуникационных систем, потокового видео, игр, банковских и других критически важных приложений. Эта база позволяет расширять ваши контейнеризованные приложения в гибридных средах — от локальной инфраструктуры до мультиоблачных развертываний.

Основные компоненты

Archon

Обеспечивает расширенные API для операций управления приложениями и ресурсами. В качестве компонента управляющей плоскости `Archon` работает исключительно на `global` кластере, выступая в роли центрального интерфейса управления операциями на уровне всего кластера. Его API-слой позволяет декларативно настраивать приложения, пространства имён и инфраструктурные ресурсы по всей платформе.

Metis

Функционирует как многофункциональный контроллер внутри `business clusters`, обеспечивая критически важные операции на уровне кластера:

- **Управление webhook**: Admission webhook для валидации ресурсов, включая применение политик по соотношению ресурсов и маркировке ресурсов.
- **Синхронизация статусов**: Поддерживает согласованность между распределёнными компонентами через:
  - согласование статуса применения `Helm chart`
  - синхронизацию `Project quota`
  - обновления статуса `Application` (запись в `Application.status`)

Captain controller manager

Выступает контроллером управления жизненным циклом приложений `Helm chart`, работающим исключительно на `global cluster`. Его обязанности включают:

- **Установку чарта**: Оркестрация развертывания `Helm chart` по кластерам
- **Управление версиями**: Обеспечение бесшовных обновлений и откатов релизов `Helm chart`
- **Удаление**: Полное удаление приложения `Helm chart` и связанных ресурсов

- **Отслеживание релизов**: Поддержание состояния и истории всех развернутых релизов `Helm chart`

Icarus

Обеспечивает централизованный веб-интерфейс управления для `Container Platform`. В качестве компонента презентационного слоя `Icarus`:

- Предоставляет комплексные визуализации дашбордов для мониторинга состояния кластера
- Позволяет выполнять развертывание и управление приложениями через GUI
- **Реализует многоарендную модель управления на основе Kubernetes RBAC**:
  - Разграничивает учётные записи арендаторов через изоляцию пространств имён
  - Управляет правами доступа к ресурсам для каждого арендатора
  - Обеспечивает изоляцию представлений для каждого арендатора
- Работает исключительно на `global cluster`, выступая в качестве единой точки управления мультикластерными операциями

Основные понятия

Типы приложений

Custom Applications

Понимание пользовательских приложении

Типы нагру

Понимание параметров

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использов

Рекомендации по лучшим практикам

Устранение распространённых проблем

Расширенные сценарии использования

Понимание

Overview

Core Concepts

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Расширенные шаблоны использования

Описание е

Типы приложений

В разделе платформы **Container Platform > Applications** можно создавать следующие типы приложений:

- **Custom Application**: Custom Application представляет собой полноценное бизнес-приложение, состоящее из одного или нескольких взаимосвязанных вычислительных компонентов (таких как Workloads, например Deployments или StatefulSets), внутренних сетевых конфигураций (Services) и других нативных ресурсов Kubernetes. Этот тип приложения предлагает гибкие методы создания, поддерживая прямое редактирование через UI, оркестрацию на YAML и шаблонные развертывания, что делает его подходящим для разработки, тестирования и производственных сред. Подробнее об этом типе приложения можно узнать в разделе Custom Application. Различные типы нативных приложений можно создавать следующими способами:
  - Create from Image: Быстрое создание приложений с использованием существующих контейнерных образов.
  - Create from YAML: Создание приложений с помощью YAML-конфигураций.
  - Create from Code: Создание приложений из исходного кода.
- **Helm Chart Application**: Helm Chart Application позволяет развертывать и управлять приложениями, упакованными в Helm Charts. Helm Charts — это наборы предварительно настроенных ресурсов Kubernetes, которые можно развернуть как единое целое, упрощая установку и управление сложными приложениями. Подробнее об этом типе приложения можно узнать в разделе Helm Chart Application
- **Operator Backed Application**: Operator Backed Application использует возможности Kubernetes Operators для автоматизации управления жизненным циклом сложных приложений. Развертывая приложение, поддерживаемое Operator, вы получаете автоматическое развертывание, масштабирование, обновления и обслуживание, поскольку Operator выступает в роли интеллектуального контроллера, адаптированного под конкретное приложение. Подробнее об этом типе приложения можно узнать в разделе Operator Backed Application.

Custom Applications

Содержание

Понимание пользовательских приложений

Пользовательское приложение — это парадигма приложения, построенная на нативных ресурсах Kubernetes (например, Deployment, Service, ConfigMap), строго соответствующая принципам декларативного дизайна API Kubernetes. Пользователи могут определять и развертывать приложения через стандартные YAML-файлы или прямые вызовы Kubernetes API, что обеспечивает тонкий контроль над жизненным циклом приложения. Приложения, созданные из таких источников, как образы, код и YAML, классифицируются как пользовательские приложения в Alauda Container Platform. Основой дизайна является баланс между гибкостью и стандартизацией, что идеально подходит для сценариев, требующих глубокой кастомизации управления.

Основные возможности

1. **Управление на основе декларативного API**

- Объединяет распределённые ресурсы (например, Deployment, Service, Ingress) в логическую единицу приложения через Application CRD, обеспечивая атомарные операции.

2. **Абстракция на уровне приложения и агрегация состояния**

- Скрывает детали низкоуровневых ресурсов (например, статус реплик Pod). Разработчики могут напрямую через ресурс Application отслеживать общее состояние приложения (например, соотношение готовых endpoint, согласованность версий).
- Поддерживает декларации зависимостей между компонентами (например, сервис базы данных должен запускаться до сервиса приложения) для обеспечения порядка и координации инициализации ресурсов.

3. **Полное управление жизненным циклом**

- Контроль версий: отслеживает исторические конфигурации, позволяя одним кликом откатиться к любой стабильной версии.
- Разрешение зависимостей: автоматически выявляет и управляет совместимостью версий между компонентами (например, соответствие версий API Service и контроллеров Ingress).

4. **Расширенная наблюдаемость**

- Агрегирует метрики состояния всех связанных ресурсов (например, доступные реплики Deployment, нагрузка на Service), предоставляя глобальный обзор через единый Dashboard.

Ценность дизайна

Измерение	Ценностное предложение
Управление сложностью	Инкапсулирует разбросанные ресурсы (например, Deployment, Service) в единую логическую сущность, снижая когнитивную и операционную нагрузку.

Измерение	Ценностное предложение
Стандартизация	Унифицирует стандарты описания приложений через Application CRD, устраняя хаос управления, вызванный фрагментацией YAML.
Совместимость с экосистемой	Бесшовно интегрируется с нативными инструментами (например, kubectl, Kubernetes Dashboard) и поддерживает расширения Helm Chart.
Эффективность DevOps	Реализует декларативную доставку через GitOps пайплайны (например, Argo CD), ускоряя автоматизацию CI/CD.

Архитектурный дизайн CRD пользовательского приложения

Модуль пользовательского приложения определяет два основных CRD-ресурса, формирующих атомарные абстракции для управления приложением:

Измерение	Ценностное предложение
Application	Описывает метаданные и топологию компонентов логической единицы приложения, агрегируя ресурсы, такие как Deployment/Service, в единую сущность.
ApplicationHistory	Фиксирует все операции жизненного цикла приложения (создание/обновление/откат/удаление) в виде версионированных снимков, тесно связанный с Application CRD для сквозного аудита.

Определение Application CRD

Application CRD использует поле `spec.componentKinds` для объявления типов ресурсов Kubernetes (например, Deployment, Service), обеспечивая управление жизненным циклом между ресурсами.

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: applications.app.k8s.io
spec:
  group: app.k8s.io
  names:
    kind: Application
    listKind: ApplicationList
    plural: applications
    singular: application
  scope: Namespaced
  subresources:
    status: {}
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description: 'APIVersion определяет версионированную схему этог
o представления объекта. Сервера должны преобразовывать распознанные схем
ы в последнее внутреннее значение и могут отклонять нераспознанные значен
ия. Подробнее: https://github.com/kubernetes/community/blob/master/contri
butors/devel/sig-architecture/api-conventions.md#resources'
          type: string
        kind:
          description: 'Kind — строковое значение, представляющее REST-ре
сурс, который представляет этот объект. Сервера могут выводить это из кон
ечной точки, на которую клиент отправляет запросы. Не может быть обновлен
о. В CamelCase. Подробнее: https://github.com/kubernetes/community/blob/m
aster/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
          type: string
        metadata:
          description: 'Metadata — объект, представляющий метаданные ресу
рса Kubernetes. Подробнее: https://github.com/kubernetes/community/blob/m
aster/contributors/devel/sig-architecture/api-conventions.md#metadata'
          type: object
        spec:
          properties:
            assemblyPhase:
              description: |
                Инсталлятор может установить это поле, чтобы указать, что
компоненты приложения
                всё ещё разворачиваются ("Pending") или все уже развернут
```

```
ы ("Succeeded"). Если
                приложение не может быть успешно собрано, инсталлятор мож
ет установить это поле в "Failed".'
              type: string
            componentKinds:
              description: |
                Этот массив GroupKinds используется для указания типов ре
сурсов, из которых
                состоит приложение. Например, приложение, имеющее сервис
и деплоймент,
                установит это поле в [{"group":"core","kind": "Service"},
{"group":"apps","kind":"Deployment"}]
              items:
                description: 'Элемент GroupKinds, структура вида "{\"grou
p\":\"core\",\"kind\": \"Service\"}"'
                type: object
              type: array
            descriptor:
              properties:
                description:
                  description: 'Краткое, удобочитаемое текстовое описание
приложения.'
                  type: string
                icons:
                  description: 'Список иконок приложения. Информация об и
конках включает источник, размер и MIME-тип.'
                  items:
                    properties:
                      size:
                        description: 'Размер иконки.'
                        type: string
                      src:
                        description: 'Источник иконки.'
                        type: string
                      type:
                        description: 'MIME-тип иконки.'
                        type: string
                    required:
                    - src
                    type: object
                  type: array
                keywords:
                  description: 'Список ключевых слов, идентифицирующих пр
иложение.'
```

```
                items:
                  type: string
                type: array
              links:
                description: 'Ссылки — список описательных URL, предназ
наченных для предоставления дополнительной документации, дашбордов и т.
д.'
                items:
                  properties:
                    description:
                      description: 'Описание ссылки.'
                      type: string
                    url:
                      description: 'URL ссылки.'
                      type: string
                  type: object
                type: array
              maintainers:
                description: 'Список сопровождающих приложение. Каждый
сопровождающий имеет имя, email и URL. Это поле предназначено для дистриб
ьюторов приложения для указания своей идентичности и контактной информаци
и.'
                items:
                  properties:
                    email:
                      description: 'Email сопровождающего.'
                      type: string
                    name:
                      description: 'Имя сопровождающего.'
                      type: string
                    url:
                      description: 'URL для связи с сопровождающим.'
                      type: string
                  type: object
                type: array
              notes:
                description: 'Заметки содержат удобочитаемые фрагменты,
предназначенные для быстрого старта пользователей приложения. Могут быть
в виде простого текста или CommonMark markdown.'
                type: string
              owners:
                items:
                  properties:
                    email:
```

```
                      description: 'Email владельца.'
                      type: string
                    name:
                      description: 'Имя владельца.'
                      type: string
                    url:
                      description: 'URL для связи с владельцем.'
                      type: string
                  type: object
                type: array
              type:
                description: 'Тип приложения (например, WordPress, MySQ
L, Cassandra). В одном namespace может быть много приложений с разными им
енами. Поле type используется для указания, что они все одного типа прило
жения.'
                type: string
              version:
                description: 'Индикатор версии приложения (например, 5.
7 для MySQL версии 5.7).'
                type: string
            type: object
          info:
            description: 'Info содержит удобочитаемые пары ключ-значени
е для приложения.'
            items:
              properties:
                name:
                  description: 'Имя информации.'
                  type: string
                type:
                  description: 'Тип информации.'
                  type: string
                value:
                  description: 'Значение информации.'
                  type: string
                valueFrom:
                  description: 'Ссылка на значение из другого ресурса.'
                  properties:
                    configMapKeyRef:
                      description: 'Ссылка на ключ ConfigMap.'
                      properties:
                        key:
                          type: string
                      type: object
```

```
                    ingressRef:
                      description: 'Ссылка на ingress.'
                      properties:
                        host:
                          description: 'Хост ссылки ingress.'
                          type: string
                        path:
                          description: 'Путь ссылки ingress.'
                          type: string
                      type: object
                    secretKeyRef:
                      description: 'Ссылка на секретный ключ.'
                      properties:
                        key:
                          type: string
                      type: object
                    serviceRef:
                      description: 'Ссылка на сервис.'
                      properties:
                        path:
                          description: 'Путь ссылки сервиса.'
                          type: string
                        port:
                          description: 'Порт ссылки сервиса.'
                          format: int32
                          type: integer
                      type: object
                    type:
                      type: string
                  type: object
                type: object
              type: array
            selector:
              description: 'Селектор используется для сопоставления ресур
сов, принадлежащих приложению. Все ресурсы приложения должны иметь метки,
чтобы соответствовать этому селектору. Пользователи должны использовать м
етку app.kubernetes.io/name на всех компонентах приложения и установить с
електор для соответствия этой метке. Например, {"matchLabels": [{"app.kub
ernetes.io/name": "my-cool-app"}]} должен использоваться как селектор для
приложения с именем "my-cool-app", и каждый компонент должен содержать со
ответствующую метку.'
              type: object
          type: object
        status:
```

```
          description: 'Статус суммирует текущее состояние объекта.'
          properties:
            observedGeneration:
              description: 'observedGeneration — поколение, недавно наблю
даемое компонентом, ответственным за действия при изменениях желаемого со
стояния ресурса.'
              format: int64
              type: integer
          type: object
  version: v1beta1
  versions:
  - name: v1beta1
    served: true
    storage: true
```

Определение ApplicationHistory

ApplicationHistory CRD фиксирует все операции жизненного цикла (например, создание, обновление, откат) в виде версионированных снимков и тесно интегрирован с Application CRD для обеспечения сквозного аудита.

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
```

Типы нагрузок

Помимо создания облачно-нативных приложений через модуль Applications, нагрузки также можно создавать напрямую в Container Platform > Workloads:

- Deployment: Наиболее часто используемый контроллер нагрузки для развертывания безсостоящих приложений. Обеспечивает запуск заданного количества реплик Pod, поддерживает поэтапные обновления и откаты, идеально подходит для безсостоящих сервисов, таких как веб-серверы и API.
- DaemonSet: Обеспечивает запуск Pod на каждом узле (или на определённых узлах) кластера. Pods автоматически создаются при добавлении узлов и удаляются при их удалении. Идеально подходит для задач на уровне узла, таких как агенты логирования и демоны мониторинга.
- StatefulSet: Контроллер нагрузки для управления состоянием приложений. Обеспечивает стабильные сетевые идентификаторы (hostname) и постоянное хранилище для каждого Pod, гарантируя согласованность данных даже при перераспределении. Подходит для баз данных, распределённых кэшей и других сервисов с состоянием.
- CronJob: Управляет задачами, основанными на времени, используя cron-выражения. Система автоматически создаёт Jobs по расписанию, идеально подходит для периодических задач, таких как резервное копирование, генерация отчётов и очистка.
- Job: Нагрузка для выполнения конечных задач. Создаёт один или несколько Pods и обеспечивает заданное количество успешных завершений перед остановкой. Подходит для пакетной обработки, миграции данных и других одноразовых операций.

Помимо создания нагрузок через веб-консоль, Kubernetes также поддерживает прямое управление ресурсами более низкого уровня через CLI-инструменты::

- Pod: Наименьшая единица развертывания в Kubernetes. Pod может содержать один или несколько тесно связанных контейнеров, которые разделяют хранилище, сеть и

жизненный цикл. Pods обычно управляются контроллерами более высокого уровня (например, Deployments).

- Container: Стандартизированная единица, инкапсулирующая код приложения и зависимости, обеспечивая единообразное выполнение в разных средах. Контейнеры запускаются внутри Pods и используют ресурсы Pod.

Понимание параметров

Содержание

Overview

Параметры в Kubernetes — это аргументы командной строки, передаваемые контейнерам во время выполнения. Они соответствуют полю `args` в спецификациях Pod Kubernetes и переопределяют стандартные аргументы CMD, определённые в образах контейнеров. Параметры предоставляют гибкий способ настройки поведения приложений без необходимости пересборки образов.

Core Concepts

Что такое параметры?

Параметры — это аргументы во время выполнения, которые:

- Переопределяют стандартную инструкцию CMD в образах контейнеров
- Передаются основному процессу контейнера в виде аргументов командной строки
- Позволяют динамически настраивать поведение приложения
- Обеспечивают повторное использование одного и того же образа с разными конфигурациями

Взаимосвязь с образом контейнера

В терминологии образов контейнеров:

- **ENTRYPOINT**: Определяет исполняемый файл (соответствует `command` в Kubernetes)
- **CMD**: Задаёт аргументы по умолчанию (соответствует `args` в Kubernetes)

- **Параметры**: Переопределяют аргументы CMD при сохранении ENTRYPOINT

```
# Пример Dockerfile
FROM nginx:alpine
ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

```
# Переопределение в Kubernetes
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: nginx
    image: nginx:alpine
    args: ["-g", "daemon off;", "-c", "/custom/nginx.conf"]
```

Сценарии использования

1. Конфигурация приложения

Передача опций конфигурации приложения:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        args:
        - "--port=8080"
        - "--log-level=info"
        - "--config=/etc/app/config.yaml"
```

2. Развёртывание для разных окружений

Разные параметры для разных окружений:

```
# Development
args: ["--debug", "--reload", "--port=3000"]

# Production
args: ["--optimize", "--port=80", "--workers=4"]
```

3. Конфигурация подключения к базе данных

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: db-client
    image: postgres:13
    args:
    - "psql"
    - "-h"
    - "postgres.example.com"
    - "-p"
    - "5432"
    - "-U"
    - "myuser"
    - "-d"
    - "mydb"
```

Примеры CLI и практическое использование

Использование kubectl run

```
# Базовая передача параметров
kubectl run nginx --image=nginx:alpine --restart=Never -- -g "daemon of
f;" -c "/custom/nginx.conf"

# Несколько параметров
kubectl run myapp --image=myapp:latest --restart=Never -- --port=8080 --l
og-level=debug

# Интерактивная отладка
kubectl run debug --image=ubuntu:20.04 --restart=Never -it -- /bin/bash
```

Использование kubectl create

```
# Создание deployment с параметрами
kubectl create deployment web --image=nginx:alpine --dry-run=client -o ya
ml > deployment.yaml

# Отредактируйте сгенерированный YAML, добавив args:
# spec:
#   template:
#     spec:
#       containers:
#       - name: nginx
#         image: nginx:alpine
#         args: ["-g", "daemon off;", "-c", "/custom/nginx.conf"]

kubectl apply -f deployment.yaml
```

Сложные примеры параметров

Веб-сервер с пользовательской конфигурацией

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-custom
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-custom
  template:
    metadata:
      labels:
        app: nginx-custom
    spec:
      containers:
      - name: nginx
        image: nginx:1.21-alpine
        args:
        - "-g"
        - "daemon off;"
        - "-c"
        - "/etc/nginx/custom.conf"
        ports:
        - containerPort: 80
        volumeMounts:
        - name: config
          mountPath: /etc/nginx/custom.conf
          subPath: nginx.conf
      volumes:
      - name: config
        configMap:
          name: nginx-config
```

Приложение с множеством параметров

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: app
    image: mycompany/myapp:v1.2.3
    args:
    - "--server-port=8080"
    - "--database-url=postgresql://db:5432/mydb"
    - "--log-level=info"
    - "--metrics-enabled=true"
    - "--cache-size=256MB"
    - "--worker-threads=4"
```

Рекомендации по лучшим практикам

1. Принципы проектирования параметров

- **Используйте осмысленные имена параметров**: `--port=8080` вместо `-p 8080`
- **Обеспечьте разумные значения по умолчанию**: чтобы приложения работали без параметров
- **Документируйте все параметры**: включайте справочную информацию и примеры
- **Проверяйте ввод**: валидируйте значения параметров и выводите сообщения об ошибках

2. Вопросы безопасности

```
# ❌ Избегайте передачи чувствительных данных в параметрах
args: ["--api-key=secret123", "--password=mypass"]

# ✅ Используйте переменные окружения для секретов
env:
- name: API_KEY
  valueFrom:
    secretKeyRef:
      name: app-secrets
      key: api-key
args: ["--config-from-env"]
```

3. Управление конфигурацией

```
# ✅ Комбинируйте параметры с ConfigMaps
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    args:
    - "--config=/etc/config/app.yaml"
    - "--log-level=info"
    volumeMounts:
    - name: config
      mountPath: /etc/config
  volumes:
  - name: config
    configMap:
      name: app-config
```

Устранение распространённых проблем

1. Параметр не распознаётся

```
# Проверьте логи контейнера
kubectl logs pod-name

# Распространённая ошибка: unknown flag
# Решение: проверьте синтаксис параметра и документацию приложения
```

2. Переопределение параметра не работает

```
# ❌ Неправильно: смешивание command и args
command: ["myapp", "--port=8080"]
args: ["--log-level=debug"]

# ✅ Правильно: используйте только args для переопределения CMD
args: ["--port=8080", "--log-level=debug"]
```

3. Отладка проблем с параметрами

```
# Запустите контейнер интерактивно для тестирования параметров
kubectl run debug --image=myapp:latest -it --rm --restart=Never -- /bin/s
h

# Внутри контейнера вручную проверьте команду
/app/myapp --port=8080 --log-level=debug
```

Расширенные сценарии использования

1. Условные параметры с Init Containers

```
apiVersion: v1
kind: Pod
spec:
  initContainers:
  - name: config-generator
    image: busybox
    command: ['sh', '-c']
    args:
    - |
      if [ "$ENVIRONMENT" = "production" ]; then
        echo "--optimize --workers=8" > /shared/args
      else
        echo "--debug --reload" > /shared/args
      fi
    volumeMounts:
    - name: shared
      mountPath: /shared
  containers:
  - name: app
    image: myapp:latest
    command: ['sh', '-c']
    args: ['exec myapp $(cat /shared/args)']
    volumeMounts:
    - name: shared
      mountPath: /shared
  volumes:
  - name: shared
    emptyDir: {}
```

2. Шаблонизация параметров с Helm

```yaml
# values.yaml
app:
  parameters:
    port: 8080
    logLevel: info
    workers: 4

# deployment.yaml template
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        args:
        - "--port={{ .Values.app.parameters.port }}"
        - "--log-level={{ .Values.app.parameters.logLevel }}"
        - "--workers={{ .Values.app.parameters.workers }}"
```

Параметры предоставляют мощный механизм настройки контейнеризованных приложений в Kubernetes. Понимая, как правильно использовать параметры, вы можете создавать гибкие, переиспользуемые и удобные в сопровождении развёртывания, адаптирующиеся к разным окружениям и требованиям.

Понимание переменных окружения

Содержание

Overview

Переменные окружения в Kubernetes — это пары ключ-значение, которые предоставляют конфигурационные данные контейнерам во время выполнения. Они обеспечивают гибкий и безопасный способ внедрения конфигурационной информации, секретов и параметров выполнения в ваши приложения без необходимости изменять образы контейнеров или код приложений.

Core Concepts

Что такое переменные окружения?

Переменные окружения — это:

- пары ключ-значение, доступные процессам, работающим внутри контейнеров
- механизм конфигурации во время выполнения, не требующий пересборки образов
- стандартный способ передачи конфигурационных данных приложениям
- доступные через стандартные API операционной системы на любом языке программирования

Источники переменных окружения в Kubernetes

Kubernetes поддерживает несколько источников переменных окружения:

Тип источника	Описание	Сценарий использования
Статические значения	Прямые пары ключ-значение	Простая конфигурация
ConfigMap	Ссылка на ключи ConfigMap	Неконфиденциальная конфигурация

Тип источника	Описание	Сценарий использования
Secret	Ссылка на ключи Secret	Конфиденциальные данные (пароли, токены)
Field Reference	Метаданные Pod/Container	Динамическая информация во время выполнения
Resource Reference	Запросы/лимиты ресурсов	Конфигурация с учётом ресурсов

Приоритет переменных окружения

Переменные окружения переопределяют конфигурацию в следующем порядке:

1. **Kubernetes env** (наивысший приоритет)
2. **Ссылки на ConfigMaps/Secrets**
3. **Инструкции ENV в Dockerfile**
4. **Значения по умолчанию приложения** (наименьший приоритет)

Сценарии использования

1. Конфигурация приложения

Основные настройки приложения:

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: web-app
    image: myapp:latest
    env:
    - name: PORT
      value: "8080"
    - name: LOG_LEVEL
      value: "info"
    - name: ENVIRONMENT
      value: "production"
    - name: MAX_CONNECTIONS
      value: "100"
```

2. Конфигурация базы данных

Настройки подключения к базе данных с использованием ConfigMaps и Secrets:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  DB_HOST: "postgres.example.com"
  DB_PORT: "5432"
  DB_NAME: "myapp"
  DB_POOL_SIZE: "10"

---
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  DB_USER: bXl1c2Vy  # base64 encoded "myuser"
  DB_PASSWORD: bXlwYXNzd29yZA==  # base64 encoded "mypassword"

---
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    env:
    # Из ConfigMap
    - name: DB_HOST
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_HOST
    - name: DB_PORT
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_PORT
    - name: DB_NAME
      valueFrom:
        configMapKeyRef:
          name: db-config
```

```
        key: DB_NAME
  # Из Secret
  - name: DB_USER
    valueFrom:
      secretKeyRef:
        name: db-secret
        key: DB_USER
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: db-secret
        key: DB_PASSWORD
```

3. Динамическая информация во время выполнения

Доступ к метаданным Pod и Node:

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    env:
    # Информация о Pod
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    # Информация о ресурсах
    - name: CPU_REQUEST
      valueFrom:
        resourceFieldRef:
          resource: requests.cpu
    - name: MEMORY_LIMIT
      valueFrom:
        resourceFieldRef:
          resource: limits.memory
```

4. Конфигурация для разных окружений

Различные конфигурации для разных сред:

```
# Среда разработки
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-dev
data:
  DEBUG: "true"
  LOG_LEVEL: "debug"
  CACHE_TTL: "60"
  RATE_LIMIT: "1000"

---
# Продакшен среда
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-prod
data:
  DEBUG: "false"
  LOG_LEVEL: "warn"
  CACHE_TTL: "3600"
  RATE_LIMIT: "100"

---
# Деплой с использованием конфигурации для конкретного окружения
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        envFrom:
        - configMapRef:
            name: app-config-prod  # Для разработки заменить на app-confi
g-dev
```

Примеры CLI и практическое использование

Использование kubectl run

```
# Установка переменных окружения напрямую
kubectl run myapp --image=nginx --env="PORT=8080" --env="DEBUG=true"

# Несколько переменных окружения
kubectl run webapp --image=myapp:latest \
  --env="DATABASE_URL=postgresql://localhost:5432/mydb" \
  --env="REDIS_URL=redis://localhost:6379" \
  --env="LOG_LEVEL=info"

# Интерактивный pod с переменными окружения
kubectl run debug --image=ubuntu:20.04 -it --rm \
  --env="TEST_VAR=hello" \
  --env="ANOTHER_VAR=world" \
  -- /bin/bash
```

Использование kubectl create

```
# Создание ConfigMap из литеральных значений
kubectl create configmap app-config \
  --from-literal=DATABASE_HOST=postgres.example.com \
  --from-literal=DATABASE_PORT=5432 \
  --from-literal=CACHE_SIZE=256MB

# Создание ConfigMap из файла
echo "DEBUG=true" > app.env
echo "LOG_LEVEL=debug" >> app.env
kubectl create configmap app-env --from-env-file=app.env

# Создание Secret для конфиденциальных данных
kubectl create secret generic db-secret \
  --from-literal=username=myuser \
  --from-literal=password=mypassword
```

Сложные примеры переменных окружения

Микросервисы с сервис-дискавери

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: service-config
data:
  USER_SERVICE_URL: "http://user-service:8080"
  ORDER_SERVICE_URL: "http://order-service:8080"
  PAYMENT_SERVICE_URL: "http://payment-service:8080"
  NOTIFICATION_SERVICE_URL: "http://notification-service:8080"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
spec:
  template:
    spec:
      containers:
      - name: gateway
        image: api-gateway:latest
        env:
        - name: PORT
          value: "8080"
        - name: ENVIRONMENT
          value: "production"
        envFrom:
        - configMapRef:
            name: service-config
        - secretRef:
            name: api-keys
```

Мультиконтейнерный Pod с общей конфигурацией

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-app
spec:
  containers:
  # Основное приложение
  - name: app
    image: myapp:latest
    env:
    - name: ROLE
      value: "primary"
    - name: SHARED_SECRET
      valueFrom:
        secretKeyRef:
          name: shared-secret
          key: token
    envFrom:
    - configMapRef:
        name: shared-config

  # Sidecar контейнер
  - name: sidecar
    image: sidecar:latest
    env:
    - name: ROLE
      value: "sidecar"
    - name: MAIN_APP_URL
      value: "http://localhost:8080"
    - name: SHARED_SECRET
      valueFrom:
        secretKeyRef:
          name: shared-secret
          key: token
    envFrom:
    - configMapRef:
        name: shared-config
```

Лучшие практики

1. Лучшие практики безопасности

```yaml
# ✅ Используйте Secrets для конфиденциальных данных
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
type: Opaque
data:
  api-key: <base64-encoded-value>
  database-password: <base64-encoded-value>

---
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    env:
    # ✅ Ссылка на секреты
    - name: API_KEY
      valueFrom:
        secretKeyRef:
          name: app-secrets
          key: api-key
    # ❌ Избегайте хардкода конфиденциальных данных
    # - name: API_KEY
    #   value: "secret-api-key-123"
```

2. Организация конфигурации

```
# ✅ Организуйте конфигурацию по назначению
apiVersion: v1
kind: ConfigMap
metadata:
  name: database-config
data:
  DB_HOST: "postgres.example.com"
  DB_PORT: "5432"
  DB_POOL_SIZE: "10"

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cache-config
data:
  REDIS_HOST: "redis.example.com"
  REDIS_PORT: "6379"
  CACHE_TTL: "3600"

---
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    envFrom:
    - configMapRef:
        name: database-config
    - configMapRef:
        name: cache-config
```

3. Именование переменных окружения

```
# ✅ Используйте единообразные соглашения об именах
env:
- name: DATABASE_HOST      # Чёткие, описательные имена
  value: "postgres.example.com"
- name: DATABASE_PORT      # Используйте подчёркивания для разделения
  value: "5432"
- name: LOG_LEVEL          # Используйте заглавные буквы для переменных о
кружения
  value: "info"
- name: FEATURE_FLAG_NEW_UI  # Префиксы для связанных переменных
  value: "true"

# ❌ Избегайте неясных или непоследовательных имён
# - name: db               # Слишком короткое
# - name: databaseHost     # Несогласованное написание
# - name: log-level        # Несогласованный разделитель
```

4. Значения по умолчанию и валидация

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    env:
    - name: PORT
      value: "8080"          # Предоставляйте разумные значения по умолча
нию
    - name: LOG_LEVEL
      value: "info"          # Безопасные значения по умолчанию
    - name: TIMEOUT_SECONDS
      value: "30"            # Указывайте единицы измерения в именах
    - name: MAX_RETRIES
      value: "3"             # Ограничивайте количество повторных попыток
```

Понимание команд запуска

Содержание

1. Init-контейнеры с пользовательскими командами
2. Sidecar-контейнеры с разными командами
3. Шаблоны Job с пользовательскими командами

Overview

Команды запуска в Kubernetes определяют основной исполняемый файл, который запускается при старте контейнера. Они соответствуют полю `command` в спецификациях Pod Kubernetes и переопределяют стандартную инструкцию ENTRYPOINT, заданную в образах контейнеров. Команды запуска обеспечивают полный контроль над процессом, который выполняется внутри ваших контейнеров.

Core Concepts

Что такое команды запуска?

Команды запуска — это:

- Основной исполняемый файл, который запускается при старте контейнера
- Переопределяют инструкцию ENTRYPOINT в образах контейнеров
- Определяют главный процесс (PID 1) внутри контейнера
- Работают совместно с параметрами (args), формируя полную командную строку

Взаимосвязь с Dockerfile и параметрами

Понимание взаимосвязи между инструкциями Dockerfile и полями Kubernetes:

Dockerfile	Kubernetes	Назначение
ENTRYPOINT	`command`	Определяет исполняемый файл

Dockerfile	Kubernetes	Назначение
CMD	`args`	Задает аргументы по умолчанию

```
# Пример Dockerfile
FROM ubuntu:20.04
ENTRYPOINT ["/usr/bin/myapp"]
CMD ["--config=/etc/default.conf"]
```

```
# Переопределение в Kubernetes
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: myapp
    image: myapp:latest
    command: ["/usr/bin/myapp"]
    args: ["--config=/etc/custom.conf", "--debug"]
```

Взаимодействие command и args

Сценарий	Образ контейнера	Спецификация Kubernetes	Итоговая команда
По умолчанию	ENTRYPOINT + CMD	(отсутствует)	ENTRYPOINT + CMD
Переопределение только args	ENTRYPOINT + CMD	`args: ["new-args"]`	ENTRYPOINT + new-args
Переопределение только command	ENTRYPOINT + CMD	`command: ["new-cmd"]`	new-cmd
Переопределение обоих	ENTRYPOINT + CMD	`command: ["new-cmd"]` `args: ["new-args"]`	new-cmd + new-args

Сценарии использования

1. Пользовательский запуск приложения

Запуск разных приложений с использованием одного базового образа:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
  - name: nginx
    image: ubuntu:20.04
    command: ["/usr/sbin/nginx"]
    args: ["-g", "daemon off;", "-c", "/etc/nginx/nginx.conf"]
```

2. Отладка и устранение неполадок

Переопределение стандартной команды для запуска оболочки с целью отладки:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: debug-pod
spec:
  containers:
  - name: debug
    image: myapp:latest
    command: ["/bin/bash"]
    args: ["-c", "sleep 3600"]
```

3. Скрипты инициализации

Запуск пользовательской инициализации перед стартом основного приложения:

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    command: ["/bin/sh"]
    args:
    - "-c"
    - |
      echo "Инициализация приложения..."
      /scripts/init.sh
      echo "Запуск основного приложения..."
      exec /usr/bin/myapp --config=/etc/app.conf
```

4. Многоцелевые образы

Использование одного образа для разных задач:

```yaml
# Веб-сервер
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  template:
    spec:
      containers:
      - name: web
        image: myapp:latest
        command: ["/usr/bin/myapp"]
        args: ["server", "--port=8080"]

---
# Фоновый воркер
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker
spec:
  template:
    spec:
      containers:
      - name: worker
        image: myapp:latest
        command: ["/usr/bin/myapp"]
        args: ["worker", "--queue=tasks"]

---
# Миграция базы данных
apiVersion: batch/v1
kind: Job
metadata:
  name: migrate
spec:
  template:
    spec:
      containers:
      - name: migrate
        image: myapp:latest
        command: ["/usr/bin/myapp"]
        args: ["migrate", "--up"]
```

```
      restartPolicy: Never
```

Примеры CLI и практическое использование

Использование kubectl run

```
# Полное переопределение команды
kubectl run debug --image=nginx:alpine --command -- /bin/sh -c "sleep 360
0"

# Запуск интерактивной оболочки
kubectl run -it debug --image=ubuntu:20.04 --restart=Never --command -- /
bin/bash

# Пользовательский запуск приложения
kubectl run myapp --image=myapp:latest --command -- /usr/local/bin/start.
sh --config=/etc/app.conf

# Одноразовая задача
kubectl run task --image=busybox --restart=Never --command -- /bin/sh -c
"echo 'Task completed'"
```

Использование kubectl create job

```
# Создание job с пользовательской командой
kubectl create job backup --image=postgres:13 --dry-run=client -o yaml --
pg_dump -h db.example.com mydb > backup.yaml

# Применение job
kubectl apply -f backup.yaml
```

Сложные примеры команд запуска

Многоэтапная инициализация

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: complex-init
spec:
  containers:
  - name: app
    image: myapp:latest
    command: ["/bin/bash"]
    args:
    - "-c"
    - |
      set -e
      echo "Шаг 1: Проверка зависимостей..."
      /scripts/check-deps.sh

      echo "Шаг 2: Настройка конфигурации..."
      /scripts/setup-config.sh

      echo "Шаг 3: Выполнение миграций базы данных..."
      /scripts/migrate.sh

      echo "Шаг 4: Запуск приложения..."
      exec /usr/bin/myapp --config=/etc/app/config.yaml
    volumeMounts:
    - name: scripts
      mountPath: /scripts
    - name: config
      mountPath: /etc/app
  volumes:
  - name: scripts
    configMap:
      name: init-scripts
      defaultMode: 0755
  - name: config
    configMap:
      name: app-config
```

Условная логика запуска

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: conditional-app
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        command: ["/bin/sh"]
        args:
        - "-c"
        - |
          if [ "$APP_MODE" = "worker" ]; then
            exec /usr/bin/myapp worker --queue=$QUEUE_NAME
          elif [ "$APP_MODE" = "scheduler" ]; then
            exec /usr/bin/myapp scheduler --interval=60
          else
            exec /usr/bin/myapp server --port=8080
          fi
        env:
        - name: APP_MODE
          value: "server"
        - name: QUEUE_NAME
          value: "default"
```

Лучшие практики

1. Обработка сигналов и корректное завершение

```
# ✅ Корректная обработка сигналов
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    command: ["/bin/bash"]
    args:
    - "-c"
    - |
      # Перехват SIGTERM для корректного завершения
      trap 'echo "Получен SIGTERM, завершаемся корректно..."; kill -TERM
$PID; wait $PID' TERM

      # Запуск основного приложения в фоне
      /usr/bin/myapp --config=/etc/app.conf &
      PID=$!

      # Ожидание процесса
      wait $PID
```

2. Обработка ошибок и логирование

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    command: ["/bin/bash"]
    args:
    - "-c"
    - |
      set -euo pipefail  # Выход при ошибках, неопределённых переменных,
ошибках пайпа

      log() {
        echo "[$(date '+%Y-%m-%d %H:%M:%S')] $*" >&2
      }

      log "Запуск инициализации приложения..."

      if ! /scripts/health-check.sh; then
        log "ОШИБКА: Проверка состояния не пройдена"
        exit 1
      fi

      log "Запуск основного приложения..."
      exec /usr/bin/myapp --config=/etc/app.conf
```

3. Вопросы безопасности

```yaml
# ✅ Запуск от имени непривилегированного пользователя
apiVersion: v1
kind: Pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
  containers:
  - name: app
    image: myapp:latest
    command: ["/usr/bin/myapp"]
    args: ["--config=/etc/app.conf"]
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
```

4. Управление ресурсами

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    command: ["/usr/bin/myapp"]
    args: ["--config=/etc/app.conf"]
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Расширенные шаблоны использования

1. Init-контейнеры с пользовательскими командами

```yaml
apiVersion: v1
kind: Pod
spec:
  initContainers:
  - name: setup
    image: busybox
    command: ["/bin/sh"]
    args:
    - "-c"
    - |
      echo "Настройка общих данных..."
      mkdir -p /shared/data
      echo "Настройка завершена" > /shared/data/status
    volumeMounts:
    - name: shared-data
      mountPath: /shared
  containers:
  - name: app
    image: myapp:latest
    command: ["/bin/sh"]
    args:
    - "-c"
    - |
      while [ ! -f /shared/data/status ]; do
        echo "Ожидание завершения настройки..."
        sleep 1
      done
      echo "Запуск приложения..."
      exec /usr/bin/myapp
    volumeMounts:
    - name: shared-data
      mountPath: /shared
  volumes:
  - name: shared-data
    emptyDir: {}
```

2. Sidecar-контейнеры с разными командами

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  # Основное приложение
  - name: app
    image: myapp:latest
    command: ["/usr/bin/myapp"]
    args: ["--config=/etc/app.conf"]

  # Sidecar для отправки логов
  - name: log-shipper
    image: fluent/fluent-bit:latest
    command: ["/fluent-bit/bin/fluent-bit"]
    args: ["--config=/fluent-bit/etc/fluent-bit.conf"]

  # Sidecar для экспорта метрик
  - name: metrics
    image: prom/node-exporter:latest
    command: ["/bin/node_exporter"]
    args: ["--path.rootfs=/host"]
```

3. Шаблоны Job с пользовательскими командами

```yaml
# Job для резервного копирования
apiVersion: batch/v1
kind: Job
metadata:
  name: database-backup
spec:
  template:
    spec:
      containers:
      - name: backup
        image: postgres:13
        command: ["/bin/bash"]
        args:
        - "-c"
        - |
          set -e
          echo "Начало резервного копирования в $(date)"
          pg_dump -h $DB_HOST -U $DB_USER $DB_NAME > /backup/dump-$(date
+%Y%m%d-%H%M%S).sql
          echo "Резервное копирование завершено в $(date)"
        env:
        - name: DB_HOST
          value: "postgres.example.com"
        - name: DB_USER
          value: "backup_user"
        - name: DB_NAME
          value: "myapp"
        volumeMounts:
        - name: backup-storage
          mountPath: /backup
      restartPolicy: Never
      volumes:
      - name: backup-storage
        persistentVolumeClaim:
          claimName: backup-pvc
```

Команды запуска обеспечивают полный контроль над выполнением контейнеров в Kubernetes. Понимая, как правильно настраивать и использовать команды запуска, вы можете создавать гибкие, поддерживаемые и надежные контейнеризированные приложения, соответствующие вашим конкретным требованиям.

Описание единиц ресурсов

- CPU: Допустимые единицы: core, m (милликор). Где 1 core = 1000 m.
- Память: Допустимые единицы: Mi (1 MiB = 2^20 байт), Gi (1 GiB = 2^30 байт). Где 1 Gi = 1024 Mi.
- Виртуальный GPU (необязательно): Этот параметр действует только при наличии GPU-ресурсов в кластере. Количество виртуальных ядер GPU; 100 виртуальных ядер равны 1 физическому ядру GPU. Поддерживаются положительные целые числа.
- Видеопамять (необязательно): Этот параметр действует только при наличии GPU-ресурсов в кластере. Видеопамять виртуального GPU; 1 единица видеопамяти равна 256 Mi. Поддерживаются положительные целые числа.

Пространства имён

Создание пространств имён

- Понимание пространств имён
- Создание пространств имён через веб-к
- Создание пространства имён в проекте
- Создание пространства имён через kub

Импорт пространств имён

- Overview
- Use Cases
- Prerequisites
- Procedure

Resource Q

- Понимание Re
- Квоты
- Квоты ресурсо

Политики безопасности Pod

Назначение UID/GID

- Включение назначения UID/GID
- Проверка назначения UID/GID

Управление участниками пространства имё

Обновление Namespaces

- Обновление Quotas
- Обновление Container LimitRanges
- Обновление Pod Security Admission

Удаление/исключение пространств имён

- Удаление пространств имён

Исключение пространств имён

Создание пространств имён

Содержание

Понимание пространств имён

Обратитесь к официальной документации Kubernetes: Namespaces ↗

> В Kubernetes пространства имён предоставляют механизм изоляции групп ресурсов внутри одного кластера. Имена ресурсов должны быть уникальными в пределах пространства имён, но не обязательно уникальными между разными пространствами имён. Область действия, основанная на пространстве имён, применяется только к объектам с пространством имён (например, Deployments, Services и т.д.), а не к объектам, охватывающим весь кластер (например, StorageClass, Nodes, PersistentVolumes и т.д.).

Создание пространств имён через веб-консоль

> В рамках кластера, связанного с проектом, создайте новое пространство имён, соответствующее доступным квотам ресурсов проекта. Новое пространство имён работает в пределах квот ресурсов, выделенных проекту (например, CPU, память), и все ресурсы в пространстве имён должны находиться в связанном кластере.

1. В представлении **Project Management** нажмите на ***Project Name***, для которого хотите создать пространство имён.
2. В левой навигационной панели выберите **Namespaces** > **Namespaces**.
3. Нажмите **Create Namespace**.
4. Настройте **Basic Information**.

Параметр	Описание
Cluster	Выберите кластер, связанный с проектом, в котором будет размещено пространство имён.
Namespace	Имя пространства имён должно содержать обязательный префикс — имя проекта.

5. (Опционально) Настройте Resource Quota.
   Каждый раз, когда для контейнера в пространстве имён задаётся ограничение ресурсов (limits) по вычислительным или хранилищным ресурсам, либо при добавлении нового Pod или PVC, это будет расходовать квоту, установленную здесь.
   ВНИМАНИЕ:

   - Квота ресурсов пространства имён наследуется от выделенной квоты проекта в кластере. Максимально допустимая квота для типа ресурса не может превышать оставшуюся доступную квоту проекта. Если доступная квота какого-либо ресурса достигает 0, создание пространства имён будет заблокировано. Обратитесь к администратору платформы для корректировки квот.
   - **Требования к настройке квоты GPU**:
     - Квоты GPU (vGPU или pGPU) можно настраивать только при наличии GPU-ресурсов в кластере.

- При использовании vGPU также можно задавать квоты по памяти.

Определения единиц GPU:

- **vGPU**: 100 виртуальных GPU-единиц (vGPU) = 1 физическое ядро GPU (pGPU).
  - Примечание: pGPU учитываются только целыми числами (например, 1 pGPU = 1 ядро = 100 vGPU).
- **Единицы памяти**:
  - 1 единица памяти = 256 MiB.
  - 1 GiB = 4 единицы памяти (1024 MiB = 4 × 256 MiB).

- **Поведение квоты по умолчанию**:
  - Если для типа ресурса квота не указана, по умолчанию она не ограничена.
  - Это означает, что пространство имён может использовать **все доступные ресурсы этого типа, выделенные проекту**, без явных ограничений.

Описание параметров квоты

Категория	Тип квоты	Значение и единица	Описание
Квота ресурсов хранения	Все	Gi	Общая запрашиваемая ёмкость хранения всех Persistent Volume Claims (PVC) в этом пространстве имён не может превышать это значение.
	Storage Class		Общая запрашиваемая ёмкость хранения всех Persistent Volume Claims (PVC), связанных с выбранным

Категория	Тип квоты	Значение и единица	Описание
			StorageClass в этом пространстве имён, не может превышать это значение. **Примечание**: Пожалуйста, заранее выделите StorageClass проекту, которому принадлежит пространство имён.
Расширенные ресурсы	Получены из конфигурационного словаря (ConfigMap); подробности см. в разделе Extended Resources Quotas description.	-	Эта категория не отображается, если отсутствует соответствующий конфигурационный словарь.
Другие квоты	Введите пользовательские квоты; правила ввода см. в разделе Other Quota description.	-	Для избежания проблем с дублированием ресурсов следующие значения не допускаются в качестве типов квот: • limits.cpu • limits.memory • requests.cpu • requests.memory • pods

Категория	Тип квоты	Значение и единица	Описание
			• cpu • memory

6. (Опционально) Настройте **Container Limit Range**; подробности см. в разделе Limit Range.
7. (Опционально) Настройте **Pod Security Admission**; подробности см. в разделе Pod Security Admission.
8. (Опционально) В разделе **More Configuration** добавьте метки и аннотации для текущего пространства имён.
   Совет: Вы можете определить атрибуты пространства имён через метки или дополнить пространство имён дополнительной информацией через аннотации; оба способа можно использовать для фильтрации и сортировки пространств имён.
9. Нажмите **Create**.

Создание пространства имён в проекте через CLI ac

Для создания пространства имён в проекте с помощью командной строки `ac` выполните следующую команду:

```
# Создать пространство имён в проекте с указанием конкретных кластеров
ac adm new-project-namespace <namespace-name> --project <project-name> --
cluster <cluster-name>
```

Создание пространства имён через kubectl

Примеры YAML файлов

example-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: example
  labels:
    pod-security.kubernetes.io/audit: baseline # Опция, для обеспечения б
езопасности рекомендуется выбрать режим baseline или restricted.
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/warn: baseline
```

example-resourcequota.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-resourcequota
  namespace: example
spec:
  hard:
    limits.cpu: '20'
    limits.memory: 20Gi
    pods: '500'
    requests.cpu: '2'
    requests.memory: 2Gi
```

example-limitrange.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: example-limitrange
  namespace: example
spec:
  limits:
    - default:
        cpu: 100m
        memory: 100Mi
      defaultRequest:
        cpu: 50m
        memory: 50Mi
      max:
        cpu: 1000m
        memory: 1000Mi
      type: Container
```

Создание через YAML файл

```
kubectl apply -f example-namespace.yaml
kubectl apply -f example-resourcequota.yaml
kubectl apply -f example-limitrange.yaml
```

Создание напрямую через командную строку

```
kubectl create namespace example
kubectl create resourcequota example-resourcequota --namespace=example --
hard=limits.cpu=20,limits.memory=20Gi,pods=500
kubectl create limitrange example-limitrange --namespace=example --defaul
t='cpu=100m,memory=100Mi' --default-request='cpu=50m,memory=50Mi' --max
='cpu=1000m,memory=1000Mi'
```

Импорт пространств имён

Содержание

Overview

Возможности управления жизненным циклом Namespace:

- Импорт Namespace между кластерами: импорт Namespace в проект централизует их управление во всех Kubernetes кластерах, предоставленных платформой. Это обеспечивает администраторам единое управление ресурсами и мониторинг в распределённых средах.

Отсоединение Namespace:

- Функция отсоединения Namespace позволяет разорвать связь Namespace с текущим проектом, сбросив его ассоциацию для последующего переназначения или очистки.
- Импорт Namespace в проект предоставляет ему возможности, эквивалентные нативно созданным Namespace на платформе. Это включает унаследованные политики на уровне проекта (например, Resource Quotas), единый мониторинг и централизованный контроль управления.

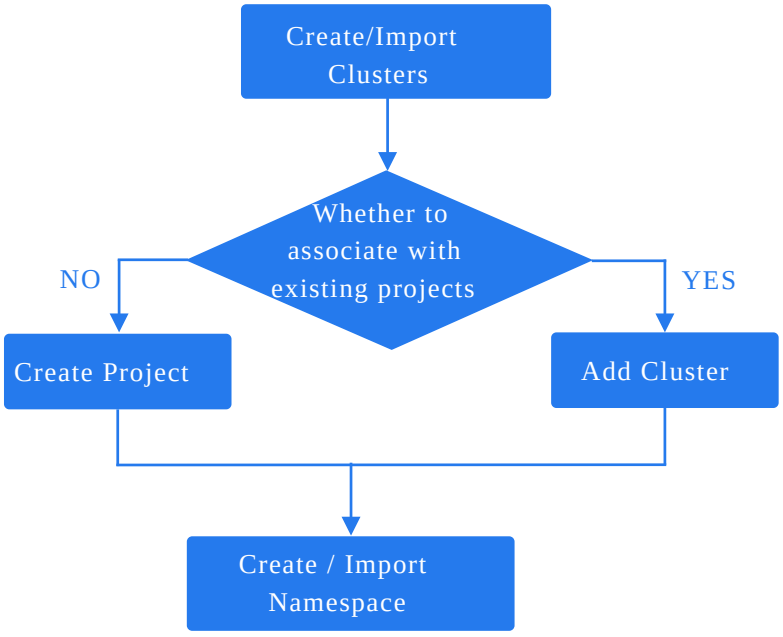
Важные замечания:

- Namespace может быть связан только с одним проектом в любой момент времени.
- Если Namespace уже связан с проектом, его нельзя импортировать или переназначить в другой проект без предварительного отсоединения от исходного проекта.

Use Cases

Типичные сценарии управления **Namespace** включают:

- При подключении нового **Kubernetes кластера** к платформе можно использовать функцию **Import Namespace** для связывания существующих **Kubernetes Namespace** с проектом. Просто выберите целевой проект и кластер для начала импорта. Это действие предоставляет **проекту** управление этими **namespace**, включая **Resource Quotas**, мониторинг и применение политик.



- **Namespace**, который был отсоединён от одного **проекта**, может быть без проблем повторно связан с другим проектом через функцию **Import Namespace** для продолжения централизованного управления.
- Namespace, которые в настоящее время не управляются никаким **проектом** (например, созданные через скрипты на уровне кластера), должны быть связаны с целевым **проектом** с помощью функции **Import Namespace** для включения

управления на уровне платформы, включая видимость и централизованное управление.

Prerequisites

- Namespace в данный момент не управляется никаким существующим проектом на платформе.
- Namespace можно импортировать только в проект, который уже связан с целевым Kubernetes кластером. Если такого проекта нет, сначала необходимо создать проект, связанный с этим кластером.

Procedure

1. В **Project Management** нажмите на ***название проекта***, в который необходимо импортировать namespace.
2. Перейдите в **Namespaces** > **Namespaces**.
3. Нажмите на кнопку **Dropdown** рядом с **Create Namespace**, затем выберите **Import Namespace**.
4. Ознакомьтесь с документацией Creating Namespaces для деталей настройки параметров.
5. Нажмите **Import**.

Resource Quota

Обратитесь к официальной документации Kubernetes: Resource Quotas ↗

Содержание

Понимание Resource Requests и Limits

Используются для ограничения ресурсов, доступных конкретному namespace. Общее использование ресурсов всеми Pod в namespace (за исключением тех, которые находятся в состоянии `Terminating`) не должно превышать квоту.

Resource Requests: Определяют минимальные ресурсы (например, CPU, память), необходимые контейнеру, помогая Kubernetes Scheduler разместить Pod на узле с достаточной емкостью.

Resource Limits: Определяют максимальные ресурсы, которые контейнер может потреблять, предотвращая исчерпание ресурсов и обеспечивая стабильность кластера.

Квоты

Resource Quotas

> Если ресурс отмечен как `Unlimited`, явная квота не применяется, но использование не может превышать доступную емкость кластера.

Resource Quotas отслеживают суммарное потребление ресурсов (например, лимиты контейнеров, новые Pod или PVC) внутри namespace.

Поддерживаемые типы квот

Поле	Описание
Resource Requests	Общие запрошенные ресурсы для всех Pod в namespace: • CPU • Память
Resource Limits	Общие лимиты ресурсов для всех Pod в namespace: • CPU • Память
Number of Pods	Максимальное количество Pod, разрешенное в namespace.

Примечание:

- Квоты namespace формируются на основе выделенных проекту ресурсов кластера. Если доступная квота по какому-либо ресурсу равна 0, создание namespace завершится ошибкой. Обратитесь к администратору.

- `Unlimited` означает, что namespace может использовать оставшиеся ресурсы проекта для данного типа ресурса.

Пример YAML файла

```yaml
# example-resourcequota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-resourcequota
  namespace: <example>
spec:
  hard:
    limits.cpu: "20"
    limits.memory: 20Gi
    pods: "500"
    requests.cpu: "2"
    requests.memory: 2Gi
```

Создание resource quota с помощью CLI

Создать через YAML файл

```
kubectl apply -f example-resourcequota.yaml
```

Создать напрямую через командную строку

```
kubectl create resourcequota example-resourcequota --namespace=<example>
--hard=limits.cpu=20,limits.memory=20Gi,pods=500
```

Storage Quotas

Типы квот:

- **All**: Общая емкость хранилища PVC в namespace.
- **Storage Class**: Общая емкость хранилища PVC для конкретного класса хранения.

Примечание: Убедитесь, что класс хранения предварительно назначен проекту, содержащему namespace.

Квоты ресурсов аппаратных ускорителей

При установке `Alauda Build of Hami` или `NVIDIA GPU Device Plugin` вы сможете использовать расширенные квоты ресурсов для аппаратных ускорителей.

См. Alauda Build of Hami и Alauda Build of NVIDIA GPU Device Plugin.

Другие квоты

Формат имен пользовательских квот должен соответствовать следующим требованиям:

- Если имя пользовательской квоты не содержит слэш (/): оно должно начинаться и заканчиваться буквой или цифрой, может содержать буквы, цифры, дефисы (-), подчеркивания (_) или точки (.), образуя квалифицированное имя длиной не более 63 символов.
- Если имя пользовательской квоты содержит слэш (/): имя делится на две части: префикс и имя, в формате: prefix/name. Префикс должен быть допустимым DNS поддоменом, а имя должно соответствовать правилам квалифицированного имени.
- DNS поддомен:
  - Метка: должна начинаться и заканчиваться строчными буквами или цифрами, может содержать дефисы (-), но не может состоять исключительно из дефисов, максимальная длина — 63 символа.
  - Поддомен: расширяет правила метки, позволяя нескольким меткам соединяться точками (.) для формирования поддомена, максимальная длина — 253 символа.

Limit Range

Содержание

Понимание Limit Range

Обратитесь к официальной документации Kubernetes: Limit Ranges ↗

Использование Kubernetes LimitRange в качестве admission controller — это **ограничения ресурсов на уровне контейнера или Pod**. Он устанавливает значения по умолчанию для запросов, лимитов и максимальных значений для контейнеров или Pod, созданных после создания или обновления LimitRange, при этом постоянно контролируя использование ресурсов контейнерами, чтобы гарантировать, что ни один ресурс не превышает определённые максимальные значения в пределах namespace.

> Запрос ресурса контейнера — это соотношение между лимитами ресурсов и overcommitment кластера. Значения запросов ресурсов служат ориентиром и критерием для планировщика при размещении контейнеров. Планировщик проверяет доступные ресурсы для каждого узла (общие ресурсы минус сумма запросов ресурсов контейнеров в Pod, запланированных на этом узле). Если суммарные

> запросы ресурсов нового контейнера Pod превышают оставшиеся доступные ресурсы узла, этот Pod не будет запланирован на данном узле.

LimitRange является admission controller:

- Он применяет значения запросов и лимитов по умолчанию для всех контейнеров, которые не задают требования к вычислительным ресурсам.
- Он отслеживает использование, чтобы убедиться, что оно не превышает максимумы ресурсов и соотношения, определённые в любом LimitRange, присутствующем в namespace.

Включает следующие настройки

Ресурс	Поле
CPU	• Default Request • Limit • Max
Memory	• Default Request • Limit • Max

Создание Limit Range с помощью CLI

Примеры YAML файлов

```yaml
# example-limitrange.yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: example-limitrange
  namespace: example
spec:
  limits:
    - default:
        cpu: 100m
        memory: 100Mi
      defaultRequest:
        cpu: 50m
        memory: 50Mi
      max:
        cpu: 1000m
        memory: 1000Mi
      type: Container
```

Создание через YAML файл

```
kubectl apply -f example-limitrange.yaml
```

Создание напрямую через командную строку

```
kubectl create limitrange example-limitrange --namespace=example --defaul
t='cpu=100m,memory=100Mi' --default-request='cpu=50m,memory=50Mi' --max
='cpu=1000m,memory=1000Mi'
```

Политики безопасности Pod

ACP поддерживает Kubernetes Pod Security Admission (PSA) и Kyverno Policy для помощи в обеспечении стандартов безопасности для Pod, работающих в ваших кластерах.

Содержание

Pod Security Admission

Обратитесь к официальной документации Kubernetes: Pod Security Admission ↗

Pod Security Admission (PSA) — это контроллер допуска Kubernetes, который обеспечивает выполнение политик безопасности на уровне namespace, проверяя спецификации Pod на соответствие предопределённым стандартам.

Режимы безопасности

PSA определяет три режима для управления обработкой нарушений политики:

Режим	Поведение	Сценарий использования
Enforce	Запрещает создание/изменение несоответствующих Pod.	Производственные среды, требующие строгого соблюдения безопасности.
Audit	Позволяет создавать Pod, но регистрирует нарушения в логах аудита.	Мониторинг и анализ инцидентов безопасности без блокировки рабочих нагрузок.
Warn	Позволяет создавать Pod, но возвращает клиенту предупреждения о нарушениях.	Тестовые среды или переходные этапы для корректировки политик.

Ключевые замечания:

- **Enforce** действует только на Pod (например, отклоняет Pod, но позволяет ресурсы, не являющиеся Pod, такие как Deployments).
- **Audit** и **Warn** применяются как к Pod, так и к их контроллерам (например, Deployments).

Стандарты безопасности

PSA определяет три стандарта безопасности для ограничения привилегий Pod:

Стандарт	Описание	Основные ограничения
Privileged	Неограниченный доступ. Подходит для доверенных	Нет проверки полей `securityContext`.

Стандарт	Описание	Основные ограничения
	рабочих нагрузок (например, системных компонентов).	
Baseline	Минимальные ограничения для предотвращения известных эскалаций привилегий.	Блокирует `hostNetwork`, `hostPID`, привилегированные контейнеры и неограниченные тома `hostPath`.
Restricted	Самая строгая политика, обеспечивающая лучшие практики безопасности.	Требует: - `runAsNonRoot: true` - `seccompProfile.type: RuntimeDefault` - Удаление Linux capabilities.

Конфигурация

Метки namespace

Применяйте метки к namespace для определения политик PSA.

Пример YAML файла

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: example-namespace
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: baseline
    pod-security.kubernetes.io/warn: baseline
```

Команда CLI

```
# Шаг 1: Обновить метки Pod Admission
kubectl label namespace <namespace-name> \
  pod-security.kubernetes.io/enforce=baseline \
  pod-security.kubernetes.io/audit=restricted \
  --overwrite

# Шаг 2: Проверить метки
kubectl get namespace <namespace-name> --show-labels
```

Исключения

Исключайте конкретных пользователей, namespace или runtime классы из проверок PSA.

Пример конфигурации:

```
apiVersion: pod-security.admission.config.k8s.io/v1
kind: PodSecurityConfiguration
exemptions:
  usernames: ['admin']
  runtimeClasses: ['nvidia']
  namespaces: ['kube-system']
```

Kyverno Policy

ACP предоставляет несколько примеров для создания Kyverno политик безопасности Pod. Примеры включают:

- **Restricted**: Restricted запрещает доступ ко всем функциям хоста и требует, чтобы Pod запускались с UID и SELinux контекстом, выделенными для namespace.
- **Restricted-v2**: Restricted-v2 запрещает доступ ко всем функциям хоста и требует, чтобы Pod запускались с UID и SELinux контекстом, выделенными для namespace. Это самая строгая политика, используемая по умолчанию для аутентифицированных пользователей. В дополнение к устаревшей политике 'restricted', она требует удаления BCEX capabilities и запрещает бинарники для повышения привилегий.

Также по умолчанию устанавливает seccomp профиль в runtime/default, если он не задан, иначе требуется этот профиль seccomp.

- **Anyuid**: Anyuid предоставляет все возможности политики restricted, но позволяет запускать Pod с любым UID и GID.
- **Hostaccess**: Hostaccess разрешает доступ ко всем namespace хоста, но требует, чтобы Pod запускались с UID и SELinux контекстом, выделенными для namespace. ВНИМАНИЕ: эта политика разрешает доступ к namespace, файловым системам и PID хоста. Используйте только для доверенных Pod. Предоставляйте с осторожностью.
- **Hostmount-anyuid**: Hostmount-anyuid предоставляет все возможности политики restricted, но разрешает хостовые монтирования и любой UID для Pod. В основном используется для recycler persistent volume. ВНИМАНИЕ: эта политика разрешает доступ к файловой системе хоста с любым UID, включая UID 0. Предоставляйте с осторожностью.
- **Hostnetwork**: Hostnetwork разрешает использование сетей и портов хоста, но требует, чтобы Pod запускались с UID и SELinux контекстом, выделенными для namespace.
- **Hostnetwork-v2**: Hostnetwork-v2 разрешает использование сетей и портов хоста, но требует, чтобы Pod запускались с UID и SELinux контекстом, выделенными для namespace. В дополнение к устаревшей политике 'hostnetwork', требует удаления BCEX capabilities и запрещает бинарники для повышения привилегий. По умолчанию устанавливает seccomp профиль в runtime/default, если он не задан, иначе требуется этот профиль seccomp.
- **Node-exporter**: Политика Node-exporter используется для Prometheus node exporter.
- **Nonroot**: Nonroot предоставляет все возможности политики restricted, но позволяет запускать Pod с любым не-root UID. Пользователь должен указать UID или он должен быть указан в манифесте контейнерного рантайма.
- **Nonroot-v2**: Nonroot-v2 предоставляет все возможности политики restricted, но позволяет запускать Pod с любым не-root UID. Пользователь должен указать UID или он должен быть указан в манифесте контейнерного рантайма. В дополнение к устаревшей политике 'nonroot', требует удаления BCEX capabilities и запрещает бинарники для повышения привилегий. По умолчанию устанавливает seccomp профиль в runtime/default, если он не задан, иначе требуется этот профиль seccomp.
- **Privileged**: Privileged разрешает доступ ко всем привилегированным функциям и функциям хоста, а также возможность запускать с любым пользователем, любой группой, любым fsGroup и с любым SELinux контекстом. ВНИМАНИЕ: это самая

либеральная политика и должна использоваться только для администрирования кластера. Предоставляйте с осторожностью.

> **NOTICE**
>
> Политика **Restricted** не равна стандарту Kubernetes Pod Security Admission 'restricted'. Возможно, вам потребуется изменить конфигурацию безопасности Pod, если вы хотите использовать Kyverno политику **Restricted** вместо стандарта Kubernetes Pod Security Admission 'restricted'.

Предварительные требования

- Установите Alauda Container Platform Compliance для Kyverno, обратитесь к документу.
- Включите feature gate `namespace-resource-manage` в настройках featuregate ACP.

Применение Kyverno политик

Веб-консоль

1. В консоли ACP перейдите в **Container Platform**, выберите namespace, в котором хотите применить политику безопасности.
2. Перейдите в **Advanced** > **Resources**.
3. Найдите ресурс типа **Policy** из группы **kyverno.io**.
4. Выберите версию "v2beta1", нажмите **Create** для создания новой Kyverno политики.
5. В диалоге **Create Resource** выберите вкладку **Samples**.
6. Выберите нужный пример политики безопасности Pod (например, `Restricted`), затем нажмите **Try**.
7. Просмотрите и при необходимости измените YAML политики, затем нажмите **Update** для применения политики.

CLI

1. Войдите в Kubernetes кластер, где хотите применить политику безопасности.
2. Выполните следующую команду для создания Kyverno политики из примерного ресурса:

```
$ kubectl get consoleyamlsamples.console.alauda.io restricted-policy -o
template --template={{.spec.yaml}}|kubectl apply -f -
$ kubectl get policies.kyverno.io
NAME         ADMISSION   BACKGROUND   READY   AGE   MESSAGE
restricted   true        true         True    1m    Ready
```

Доступные примерные ресурсы:

- restricted-policy
- restrictedv2-policy
- anyuid-policy
- hostaccess-policy
- hostmount-anyuid-policy
- hostnetwork-policy
- hostnetwork-v2-policy
- node-exporter-policy
- nonroot-policy
- nonroot-v2-policy
- privileged-policy

Назначение UID/GID

В Kubernetes каждый Pod запускается с определённым User ID (UID) и Group ID (GID) для обеспечения безопасности и правильного контроля доступа. По умолчанию Pods могут запускаться от имени пользователя root (UID 0), что может представлять угрозу безопасности. Для повышения безопасности рекомендуется назначать Pods не-root UID и GID.

ACP позволяет автоматически назначать namespace с определёнными диапазонами UID и GID, чтобы гарантировать, что все Pods в этом namespace запускаются с заданными идентификаторами пользователя и группы.

Содержание

Включение назначения UID/GID

Чтобы включить назначение UID/GID для namespace, выполните следующие шаги:

1. Перейдите в **Project Management**.
2. В левой навигационной панели нажмите **Namespace**.
3. Выберите нужный namespace.

4. Нажмите **Actions** > **Upate Pod Security Policy**.
5. Измените значение опции **Enforce** на **Restricted**, нажмите **Update**.
6. Нажмите на иконку редактирования рядом с **Labels**, добавьте метку с ключом `security.cpaas.io/enabled` и значением `true`, нажмите **Update**. (Чтобы отключить, удалите эту метку или установите значение `false`.)
7. Нажмите **Save**.

Проверка назначения UID/GID

Диапазон UID/GID

На странице с деталями namespace можно просмотреть назначенные диапазоны UID и GID в разделе **Annotations**.

Аннотация **security.cpaas.io/uid-range** указывает диапазон UID/GID, которые могут быть назначены Pods в namespace, например, **security.cpaas.io/uid-range=1000002000-1000011999** означает, что диапазон uid/gid от 1000002000 до 1000011999.

Проверка UID/GID Pod

Если pod не указывает `runAsUser` и `fsGroup` в `securityContext`, платформа автоматически назначит первое значение из назначенного диапазона uid.

1. Создайте Pod в namespace с следующим YAML-конфигом:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: uid-gid-test-pod
spec:
  containers:
  - name: test-container
    image: busybox
    command: ["sleep", "3600"]
```

2. После создания Pod получите его YAML, чтобы проверить назначенные UID и GID:

```
kubectl get pod uid-gid-test-pod -n <namespace-name> -o yaml
```

В YAML Pod будут показаны назначенные UID и GID в разделе `securityContext`:

```
apiVersion: v1
kind: Pod
metadata:
  name: uid-gid-test-pod
spec:
  containers:
  - name: test-container
    image: busybox
    command: ["sleep", "3600"]
    securityContext:
      runAsUser: 1000000
  securityContext:
    fsGroup: 1000000
```

Если pod указывает runAsUser и fsGroup в securityContext, платформа проверит, находятся ли указанные UID/GID в пределах назначенного диапазона. Если нет, создание Pod завершится с ошибкой.

1. Создайте Pod в namespace с следующим YAML-конфигом:

```
apiVersion: v1
kind: Pod
metadata:
  name: uid-gid-test-pod-invalid
spec:
  containers:
  - name: test-container
    image: busybox
    command: ["sleep", "3600"]
    securityContext:
      runAsUser: 2000000  # Неверный UID, вне назначенного диапазона
  securityContext:
    fsGroup: 2000000  # Неверный GID, вне назначенного диапазона
```

2. После применения YAML создание Pod завершится с ошибкой, указывающей, что указанные UID/GID находятся вне назначенного диапазона.

Коэффициент Overcommit

Содержание

Понимание коэффициента overcommit ресурсов в Namespace

Alauda Container Platform позволяет задать коэффициент overcommit ресурсов (CPU и памяти) для каждого namespace. Это управляет соотношением между лимитами контейнеров (максимальное использование) и запросами (гарантированный минимум) внутри данного namespace, оптимизируя использование ресурсов.

Настраивая этот коэффициент, вы обеспечиваете, что пользовательские лимиты и запросы контейнеров остаются в разумных пределах, повышая общую эффективность использования ресурсов кластера.

Ключевые понятия

- Лимиты (Limits): Максимальное количество ресурса, которое контейнер может использовать. Превышение лимитов может привести к троттлингу (CPU) или завершению процесса (память).
- Запросы (Requests): Гарантированный минимальный ресурс, необходимый контейнеру. Kubernetes планирует размещение контейнеров на основе этих запросов.
- Коэффициент Overcommit: Limits / Requests. Эта настройка определяет допустимый диапазон данного соотношения в namespace, балансируя гарантии ресурсов и предотвращая их чрезмерное потребление.

Основные возможности

- Повышение плотности использования ресурсов и стабильности приложений в namespace за счёт установки подходящего коэффициента overcommit, управляющего балансом между лимитами и запросами ресурсов.

Пример

> Предположим, что коэффициент overcommit для namespace установлен в 2. При создании приложения и указании лимита CPU в 4c, соответствующее значение запроса CPU рассчитывается как:

CPU Request = CPU Limit / Overcommit ratio. Таким образом, запрос CPU будет 4c / 2 = 2c.

Определение CRD

```yaml
# example-namespace-overcommit.yaml
apiVersion: resource.alauda.io/v1
kind: NamespaceResourceRatio
metadata:
  namespace: example
  name: example-namespace-overcommit
spec:
  cpu: 3 # Отсутствие этого поля означает наследование коэффициента overc
ommit кластера; 0 — отсутствие ограничения.
  memory: 4 # Отсутствие этого поля означает наследование коэффициента ov
ercommit кластера; 0 — отсутствие ограничения.
status:
  clusterCPU: 2 # Коэффициент overcommit кластера
  clusterMemory: 3
```

Создание коэффициента overcommit с помощью CLI

```bash
kubectl apply -f example-namespace-overcommit.yaml
```

Создание/обновление коэффициента overcommit через веб-консоль

Позволяет настроить **коэффициент overcommit** для namespace, чтобы управлять соотношением между лимитами и запросами ресурсов. Это гарантирует, что выделение ресурсов контейнерам остаётся в заданных пределах, улучшая использование ресурсов кластера.

Меры предосторожности

Если в кластере используется виртуализация узлов (например, виртуальные узлы), перед настройкой oversubscription для виртуальных машин отключите oversubscription на уровне кластера/namespace.

Порядок действий

1. Перейдите в **Project Management** и откройте **Namespaces** > **Список Namespace**.
2. Нажмите на целевой ***Namespace name***.
3. Выберите **Actions** > **Update Overcommit**.
4. Выберите подходящий **метод настройки** коэффициента overcommit для CPU или памяти в namespace.

Параметр	Описание
Inherit from Cluster	• Namespace наследует коэффициент oversubscription кластера. • Пример: если коэффициент CPU/памяти кластера равен 4, namespace по умолчанию будет 4. • Запросы контейнера = лимит / коэффициент кластера. • Если лимит не задан, используется квота контейнера по умолчанию в namespace.
Custom	• Задать специфичный для namespace коэффициент (целое число > 1). • Пример: коэффициент кластера = 4, коэффициент namespace = 2 → запросы = лимит / 2. • Оставить пустым для отключения oversubscription в namespace.

5. Нажмите **Update**.

Примечание: Изменения применяются только к вновь создаваемым Pod. Существующие Pod сохраняют свои исходные запросы до пересоздания.

Управление участниками пространства имён

Содержание

Импорт участников

Платформа поддерживает массовый импорт участников в пространство имён с назначением ролей, таких как Namespace Administrator или Developer, для предоставления соответствующих разрешений.

Ограничения и условия

- Участников можно импортировать в пространство имён только из **Project Members** проекта, к которому принадлежит пространство имён.

- Платформа не поддерживает импорт системных администраторов по умолчанию или активного пользователя.

Предварительные условия

Для импорта пользователей в качестве участников пространства имён они должны быть предварительно добавлены в проект пространства имён.

Процедура

1. В разделе **Project Management** нажмите на ***Project Name***, в котором находятся участники для импорта.
2. Перейдите в **Namespaces** > **Namespaces**.
3. Нажмите на ***Namespace Name***, в которое нужно импортировать участников.
4. Во вкладке **Namespace Members** нажмите **Import Members**.
5. Следуйте инструкциям ниже, чтобы импортировать всех или некоторых пользователей из списка в пространство имён.

> **TIP**
>
> Вы можете выбрать группу пользователей с помощью выпадающего списка в правом верхнем углу диалогового окна и выполнить нечеткий поиск, введя имя пользователя в поле поиска по имени пользователя.

- Импортировать всех пользователей из списка как участников пространства имён и массово назначить им роли.

  1. Нажмите на выпадающий список справа от пункта **Set Role** внизу диалогового окна и выберите роль для назначения.
  2. Нажмите **Import All**.

- Импортировать одного или нескольких пользователей из списка как участников пространства имён.

  1. Отметьте чекбокс перед именем пользователя/отображаемым именем для выбора одного или нескольких пользователей.

2. Нажмите на выпадающий список справа от пункта **Set Role** внизу диалогового окна и выберите роль для выбранных пользователей.
3. Нажмите **Import**.

Добавление участников

Если на платформе добавлен IDP типа OICD, пользователей OIDC можно добавить в качестве участников пространства имён.

Вы можете добавить действительные OIDC-аккаунты, соответствующие требованиям ввода, в роли пространства имён и назначить пользователю соответствующие роли.

Примечание: При добавлении участников система не проверяет действительность аккаунтов. Пожалуйста, убедитесь, что добавляемые аккаунты действительны, иначе эти аккаунты не смогут успешно войти на платформу.

Действительные OIDC-аккаунты включают: действительные аккаунты в службе аутентификации OIDC, настроенной через IDP для платформы, включая как успешно вошедших в платформу, так и не вошедших.

Предварительные условия

На платформе добавлен IDP типа **OICD**.

Процедура

1. В разделе **Project Management** нажмите на ***Project Name***, в котором находится участник для добавления.
2. Перейдите в **Namespaces** > **Namespaces**.
3. Нажмите на ***Namespace Name***, в которое нужно добавить участника.
4. Во вкладке **Namespace Members** нажмите **Add Member**.
5. В поле ввода **Username** введите имя пользователя существующего аккаунта сторонней платформы, поддерживаемого платформой.
   Примечание: Пожалуйста, убедитесь, что введённое имя пользователя соответствует существующему аккаунту на сторонней платформе, иначе этот аккаунт не сможет

успешно войти на эту платформу.

6. В выпадающем списке **Role** выберите роль, которую нужно назначить этому пользователю.
7. Нажмите **Add**.
   После успешного добавления вы сможете увидеть участника в списке участников пространства имён.
   Одновременно в списке пользователей (**Platform Management > User Management**) этот пользователь также будет отображаться. До тех пор, пока пользователь не войдёт или не будет синхронизирован с платформой, источник будет `-`, и его можно удалить; после успешного входа или синхронизации платформа зафиксирует источник аккаунта и отобразит его в списке пользователей.

Удаление участников

Удалите указанных участников пространства имён и удалите их связанные роли для отзыва разрешений в пространстве имён.

Процедура

1. В разделе **Project Management** нажмите на ***Project Name***, в котором находится участник для удаления.
2. Перейдите в **Namespaces** > **Namespaces**.
3. Нажмите на ***Namespace Name***, из которого нужно удалить участника.
4. Во вкладке **Namespace Members** нажмите ⋮ справа от записи участника, которого нужно удалить, и выберите **Remove**.
5. Нажмите **Remove**.

Обновление Namespaces

Содержание

Обновление Quotas

> Resource Quota

Обновление Resource Quota через веб-консоль

1. Перейдите в **Project Management**, затем в левой боковой панели выберите **Namespaces** > **Namespace** List.
2. Нажмите на целевое ***имя namespace***.
3. Нажмите **Actions** > **Update Quota**.

4. Отрегулируйте квоты ресурсов (CPU, Memory, Pods и т.д.) и нажмите **Update**.

Обновление Resource Quota через CLI

> Resource Quota YAML file example

```
# Шаг 1: Отредактируйте квоту namespace
kubectl edit resourcequota <quota-name> -n <namespace-name>

# Шаг 2: Проверьте изменения
kubectl get resourcequota <quota-name> -n <namespace-name> -o yaml
```

Обновление Container LimitRanges

> Limit Range

Обновление LimitRange через веб-консоль

1. В разделе **Project Management** перейдите в левой боковой панели в **Namespaces** > **Namespace** List.
2. Нажмите на целевое ***имя namespace***.
3. Нажмите **Actions** > **Update Container LimitRange**.
4. Отрегулируйте диапазон лимитов контейнера (`defaultRequest` , `default` , `max`) и нажмите **Update**.

Обновление LimitRange через CLI

> Limit Range YAML file example

```
# Шаг 1: Отредактируйте LimitRange
kubectl edit limitrange <limitrange-name> -n <namespace-name>

# Шаг 2: Проверьте изменения
kubectl get limitrange <limitrange-name> -n <namespace-name> -o yaml
```

Обновление Pod Security Admission

> Pod Security Admission

Обновление Pod Security Admission через веб-консоль

1. В разделе **Project Management** перейдите в левой боковой панели в **Namespaces** > **Namespace** List.
2. Нажмите на целевое ***имя namespace***.
3. Нажмите **Actions** > **Update Pod Security Admission**.
4. Отрегулируйте стандарт безопасности (`enforce` , `audit` , `warn`) и нажмите **Update**.

Обновление Pod Security Admission через CLI

> Update Pod Security Admission CLI command

Удаление/исключение пространств имён

Вы можете либо навсегда удалить пространство имён, либо исключить его из текущего проекта.

Содержание

Удаление пространств имён

Удалить пространство имён: Навсегда удаляет пространство имён и все ресурсы в нём (например, Pods, Services, ConfigMaps). Это действие необратимо и освобождает выделенные квоты ресурсов.

```
kubectl delete namespace <namespace-name>
```

Исключение пространств имён

Исключить пространство имён: Исключение пространства имён из текущего проекта без удаления его ресурсов. Пространство имён остаётся в кластере и может быть импортировано в другие проекты через Import Namespace.

NOTE

- Эта функция доступна исключительно в Alauda Container Platform.
- Kubernetes изначально не поддерживает "исключение" пространств имён из проектов.

```
kubectl label namespace <namespace-name> cpaas.io/project- --overwrite
```

Создание приложений

Создание приложений из обр

Предварительные требования

Процедура 1 — Workloads

Процедура 2 — Services

Процедура 3 — Ingress

Операции управления приложением

Справочная информация

Создание приложений из Cha

Меры предосторожности

Предварительные требования

Процедура

Справка по анализу статуса

Создание г

Меры предосто

Предварительны

Порядок дейст

Creating applications from Operator Backed

Понимание Operator Backed Application

ть

Создание приложений с помощью CLI

Предварительные требования

Процедура

Пример

Справка

Создание приложений из образа

Содержание

Предварительные требования

Получите адрес образа. Источником образов могут быть репозитории образов, интегрированные администратором платформы через toolchain, либо репозитории образов сторонних платформ.

- В первом случае администратор обычно назначает репозиторий образов вашему проекту, и вы можете использовать образы из него. Если нужный репозиторий образов не найден, обратитесь к администратору для выделения.
- Если это репозиторий образов сторонней платформы, убедитесь, что образы можно напрямую подтягивать из него в текущем кластере.

Процедура 1 — Workloads

1. В **Container Platform** перейдите в **Applications** > **Applications** в левой боковой панели.
2. Нажмите **Create**.
3. Выберите способ создания **Create from Image**.
4. **Выберите** или **введите** образ и нажмите **Confirm**.

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, можно фильтровать образы по параметру **Already Integrated**. Название интеграционного проекта, например, образы (registry-projectname), включает имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.
>
> При использовании образов из приватного реестра необходимо настроить соответствующий секрет для image pull. Подробнее см. Add ImagePullSecrets to ServiceAccount.

6. Следуйте приведённым ниже инструкциям для настройки соответствующих параметров.

Workload 1 — Настройка базовой информации

В разделе **Workload** > **Basic Info** настройте декларативные параметры для workloads

Параметры	Описание
Model	Выберите нужный workload:

Параметры	Описание
	• **Deployment**: Подробное описание параметров см. в Creating Deployment. • **DaemonSet**: Подробное описание параметров см. в Creating DaemonSet. • **StatefulSet**: Подробное описание параметров см. в Creating StatefulSet.
Replicas	Определяет желаемое количество реплик Pod в Deployment (по умолчанию: `1`). Настраивается в зависимости от требований workload.
More > **Update Strategy**	Настройка стратегии `rollingUpdate` для обновлений без простоя: **Max surge** (`maxSurge`): • Максимальное количество Pod, превышающее желаемое число реплик во время обновления. • Принимает абсолютные значения (например, `2`) или проценты (например, `20%`). • Вычисление процентов: `ceil(current_replicas × percentage)` . • Пример: 4.1 → `5` при расчёте от 10 реплик. **Max unavailable** (`maxUnavailable`): • Максимальное количество Pod, которые могут быть временно недоступны во время обновления. • Процентные значения не могут превышать `100%` . • Вычисление процентов: `floor(current_replicas × percentage)` . • Пример: 4.9 → `4` при расчёте от 10 реплик. **Примечания**: 1. **Значения по умолчанию**: `maxSurge=1` , `maxUnavailable=1` , если явно не заданы. 2. **Неработающие Pod** (например, в состояниях `Pending` / `CrashLoopBackOff`) считаются недоступными. 3. **Одновременные ограничения**:

Параметры	Описание
	• `maxSurge` и `maxUnavailable` не могут быть одновременно `0` или `0%`. • Если процентные значения для обоих параметров равны `0`, Kubernetes принудительно устанавливает `maxUnavailable=1` для обеспечения прогресса обновления. **Пример**: Для Deployment с 10 репликами: • `maxSurge=2` → Общее количество Pod во время обновления: `10 + 2 = 12`. • `maxUnavailable=3` → Минимальное количество доступных Pod: `10 - 3 = 7`. • Это обеспечивает доступность при контролируемом развёртывании.

Workload 2 — Настройка Pod

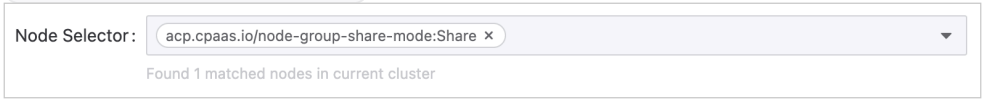
Примечание: В кластерах с разной архитектурой при развёртывании образов для одной архитектуры убедитесь в правильной настройке Node Affinity Rules для планирования Pod.

1. В разделе **Pod** настройте параметры контейнерного рантайма и управления жизненным циклом:

Параметры	Описание
Volumes	Монтирование персистентных томов в контейнеры. Поддерживаемые типы томов: `PVC`, `ConfigMap`, `Secret`, `emptyDir`, `hostPath` и др. Для деталей реализации см. Storage Volume Mounting Instructions.
Image Credential	Требуется **только** при подтягивании образов из сторонних реестров (при ручном вводе URL образа).

Параметры	Описание
	Примечание: Образы из интегрированного реестра платформы автоматически наследуют связанные секреты.
More > **Close Grace Period**	Время (по умолчанию: `30s`), отведённое Pod для корректного завершения работы после получения сигнала завершения. - В этот период Pod завершает текущие запросы и освобождает ресурсы. - Установка `0` приводит к немедленному удалению (SIGKILL), что может вызвать прерывание запросов.

2. **Node Affinity Rules**

Параметры	Описание
More > **Node Selector**	Ограничение Pod узлами с определёнными метками (например, `kubernetes.io/os: linux`). 
More > **Affinity**	Определение тонких правил планирования на основе существующих Pod. **Типы Pod Affinity**: • **Inter-Pod Affinity**: Планирование новых Pod на узлах, где размещены определённые Pod (одна топологическая область). • **Inter-Pod Anti-affinity**: Запрет совместного размещения новых Pod с определёнными Pod. **Режимы применения**: • **RequiredDuringSchedulingIgnoredDuringExecution**: Pod планируются *только* при выполнении правил. • **PreferredDuringSchedulingIgnoredDuringExecution**: Предпочтение узлам, удовлетворяющим правилам, но допускаются исключения. **Поля конфигурации**:

Параметры	Описание
	• `topologyKey`: Метка узла, определяющая топологические области (по умолчанию: `kubernetes.io/hostname`). • `labelSelector`: Фильтр целевых Pod с помощью запросов по меткам.

3. **Настройка сети**

- Kube-OVN

Параметры	Описание
Bandwidth Limits	Обеспечение QoS для сетевого трафика Pod: • **Ограничение скорости исходящего трафика**: Максимальная скорость исходящего трафика (например, `10Mbps`). • **Ограничение скорости входящего трафика**: Максимальная скорость входящего трафика.
Subnet	Назначение IP из предопределённого пула подсетей. Если не указано, используется подсеть по умолчанию для namespace.
Static IP Address	Привязка постоянных IP-адресов к Pod: • Несколько Pod из разных Deployment могут претендовать на один IP, но одновременно использовать его может только один Pod. • **Критично**: Количество статических IP должно быть ≥ количеству реплик Pod.

- Calico

Параметры	Описание
Static IP Address	Назначение фиксированных IP с жёстким ограничением уникальности: • Каждый IP может быть привязан **только к одному Pod** в кластере. • **Критично**: Количество статических IP должно быть ≥ количеству реплик Pod.

Workload 3 — Настройка контейнеров

1. В разделе **Container** настройте соответствующую информацию согласно следующим инструкциям.

Параметры	Описание
Resource Requests & Limits	• **Requests**: Минимальные CPU/память, необходимые для работы контейнера. • **Limits**: Максимальные CPU/память, разрешённые во время выполнения контейнера. Для определения единиц измерения см. Resource Units. **Коэффициент overcommit namespace**: • **Без коэффициента overcommit**: Если существуют квоты ресурсов namespace: Requests/limits контейнера наследуют значения по умолчанию namespace (можно изменить). Без квот namespace: Значения по умолчанию отсутствуют; настраивается Request вручную. • **С коэффициентом overcommit**: Requests рассчитываются автоматически как `Limits / Overcommit ratio` (неизменяемо).

Параметры	Описание
	Ограничения: • Request ≤ Limit ≤ максимальная квота namespace. • Изменение коэффициента overcommit требует пересоздания pod для вступления в силу. • Коэффициент overcommit отключает ручную настройку Request. • Отсутствие квот namespace → отсутствие ограничений ресурсов контейнера.
Extended Resources	Настройка расширенных ресурсов, доступных в кластере (например, vGPU, pGPU).
Volume Mount	Конфигурация персистентного хранилища. См. Storage Volume Mounting Instructions. **Операции**: • Для существующих томов pod: нажмите **Add** • Если томов pod нет: нажмите **Add & Mount** **Параметры**: • `mountPath`: Путь в файловой системе контейнера (например, `/data`) • `subPath`: Относительный путь файла/каталога внутри тома. Для `ConfigMap` / `Secret`: выбор конкретного ключа • `readOnly`: Монтировать только для чтения (по умолчанию: чтение-запись) См. Kubernetes Volumes↗.
Port	Открытие портов контейнера. **Пример**: Открыть TCP порт `6379` с именем `redis`. **Поля**:

Параметры	Описание
	• `protocol`: TCP/UDP • `Port`: Открываемый порт (например, `6379`) • `name`: Идентификатор, совместимый с DNS (например, `redis`)
Startup Commands & Arguments	Переопределение стандартных ENTRYPOINT/CMD: **Пример 1**: Выполнить `top -b` - **Command**: `["top", "-b"]` - **ИЛИ** Command: `["top"]`, Args: `["-b"]` **Пример 2**: Вывести `$MESSAGE`: `/bin/sh -c "while true; do echo $(MESSAGE); sleep 10; done"` См. Defining Commands↗.
More > **Environment Variables**	• Статические значения: Прямые пары ключ-значение • Динамические значения: Ссылки на ключи ConfigMap/Secret, поля pod (`fieldRef`), метрики ресурсов (`resourceFieldRef`) **Примечание**: Переменные окружения переопределяют настройки образа/конфигурационного файла.
More > **Referenced ConfigMap**	Внедрение всего ConfigMap/Secret как переменных окружения. Поддерживаемые типы Secret: `Opaque`, `kubernetes.io/basic-auth`.
More > **Health Checks**	• **Liveness Probe**: Проверка здоровья контейнера (перезапуск при сбое) • **Readiness Probe**: Проверка доступности сервиса (удаление из endpoints при сбое) См. Health Check Parameters.

Параметры	Описание
More > **Log File**	Настройка путей логов: - По умолчанию: сбор `stdout` - Паттерны файлов: например, `/var/log/*.log` **Требования**: • Драйвер хранения `overlay2`: поддерживается по умолчанию • `devicemapper`: необходимо вручную монтировать EmptyDir в каталог логов • Windows-узлы: убедитесь, что родительский каталог смонтирован (например, `c:/a` для `c:/a/b/c/*.log`)
More > **Exclude Log File**	Исключение определённых логов из сбора (например, `/var/log/aaa.log`).
More > **Execute before Stopping**	Выполнение команд перед завершением контейнера. **Пример**: `echo "stop"` **Примечание**: Время выполнения команды должно быть меньше `terminationGracePeriodSeconds` pod.

2. Нажмите **Add Container** (вверху справа) ИЛИ **Add Init Container**.
   См. Init Containers↗. Init Container:

   1. Запускается перед контейнерами приложения (последовательное выполнение).
   2. Освобождает ресурсы после завершения.
   3. Удаление разрешено, если:

      - В Pod >1 контейнера приложения И ≥1 init контейнер.
      - Не разрешено для Pod с одним контейнером приложения.

3. Нажмите **Create**.

Процедура 2 — Services

Параметры	Описание
Service	Kubernetes **Service**, обеспечивает доступ к приложению, работающему в вашем кластере, через единый внешний endpoint, даже если workload распределён по нескольким backend. Для подробного описания параметров см. Creating Services. **Примечание**: Префикс имени по умолчанию для внутреннего маршрута, создаваемого под приложением, — это имя вычислительного компонента. Если тип вычислительного компонента (режим развёртывания) — StatefulSet, рекомендуется не менять имя внутреннего маршрута (имя workload), иначе могут возникнуть проблемы с доступностью workload.

Процедура 3 — Ingress

Параметры	Описание
Ingress	Kubernetes **Ingress**, позволяет сделать ваш HTTP (или HTTPS) сетевой сервис доступным с помощью протокольно-ориентированной конфигурации, понимающей веб-концепции, такие как URI, имена хостов, пути и др. Концепция Ingress позволяет направлять трафик на разные backend в зависимости от правил, определённых через Kubernetes API. Для подробного описания параметров см. Creating Ingresses. **Примечание**: **Service**, используемый при создании **Ingress** под приложением, должен быть ресурсом, созданным в рамках текущего приложения. При этом убедитесь, что **Service** связан с workload под приложением, иначе обнаружение и доступ к workload не будут работать.

7. Нажмите **Create**.

Операции управления приложением

Для изменения конфигураций приложения используйте один из следующих способов:

1. Нажмите вертикальное многоточие (⋮) справа от списка приложений.
2. Выберите **Actions** в правом верхнем углу страницы с деталями приложения.

Операция	Описание
Update	• **Update**: Изменяет только целевой workload с использованием его определённой стратегии обновления (пример — стратегия Deployment). Сохраняет текущее количество реплик и конфигурацию развёртывания. • **Force Update**: Запускает полное развёртывание приложения с использованием стратегии обновления каждого компонента. 1. **Сценарии использования**: • Пакетные изменения конфигураций, требующие немедленного распространения по всему кластеру (например, обновления ConfigMap/Secret, используемых как переменные окружения). • Координированные перезапуски компонентов для критических обновлений безопасности. 2. **Предупреждение**: • Может вызвать временное ухудшение качества сервиса из-за массовых перезапусков. • Не рекомендуется использовать в production без проверки непрерывности бизнес-процессов. • **Сетевые последствия**: • Удаление правил Ingress: Внешний доступ остаётся доступным через `LB_IP:NodePort`, если: 1) Service типа LoadBalancer использует порты по умолчанию. 2) Сохранившиеся правила маршрутизации ссылаются на компоненты приложения. Полное прекращение внешнего доступа требует удаления Service. • Удаление Service: Необратимая потеря сетевого подключения к компонентам приложения. Связанные правила Ingress перестают работать, несмотря на сохранение объектов в API.
Delete	• **Каскадное удаление**: 1. Удаляет все дочерние ресурсы, включая Deployments, Services и

Операция	Описание
	правила Ingress. 2. Persistent Volume Claims (PVC) обрабатываются согласно политике хранения, определённой в StorageClass. • **Перед удалением**: 1. Убедитесь, что через связанные Services не идёт активный трафик. 2. Подтвердите резервное копирование данных для stateful компонентов. 3. Проверьте зависимости ресурсов с помощью `kubectl describe ownerReferences`.

Справочная информация

Инструкция по монтированию томов

Тип	Назначение
Persistent Volume Claim	Привязывает существующий PVC для запроса персистентного хранилища. **Примечание**: Выбираются только привязанные PVC (с ассоциированным PV). Непривязанные PVC приведут к ошибкам создания pod.
ConfigMap	Монтирует полные или частичные данные ConfigMap в виде файлов: • Полный ConfigMap: создаёт файлы с именами ключей под путём монтирования • Выбор подпути: монтирует конкретный ключ (например, `my.cnf`)

Тип	Назначение
Secret	Монтирует полные или частичные данные Secret в виде файлов: • Полный Secret: создаёт файлы с именами ключей под путём монтирования • Выбор подпути: монтирует конкретный ключ (например, `tls.crt`)
Ephemeral Volumes	Временный том, предоставляемый кластером, с возможностями: • Динамическое выделение • Жизненный цикл связан с pod • Поддержка декларативной конфигурации **Сценарий использования**: временное хранение данных. См. Ephemeral Volumes
Empty Directory	Временное хранилище, общее для контейнеров в одном pod: - Создаётся на узле при старте pod - Удаляется при удалении pod **Сценарий использования**: обмен файлами между контейнерами, временное хранение данных. См. EmptyDir
Host Path	Монтирует директорию хост-машины (должна начинаться с `/`, например, `/volumepath`).

Параметры проверки здоровья

Общие параметры

Параметры	Описание
Initial Delay	Время ожидания (в секундах) перед началом проверок. По умолчанию: `300`.
Period	Интервал между проверками (1-120 с). По умолчанию: `60`.

Параметры	Описание
Timeout	Время ожидания ответа проверки (1-300 с). По умолчанию: `30`.
Success Threshold	Минимальное количество последовательных успешных проверок для признания контейнера здоровым. По умолчанию: `0`.
Failure Threshold	Максимальное количество последовательных неудач для запуска действия: - `0`: отключение действий при ошибках - По умолчанию: `5` неудач → перезапуск контейнера.

Параметры, специфичные для протоколов

Параметр	Применимые протоколы	Описание
Protocol	HTTP/HTTPS	Протокол проверки здоровья
Port	HTTP/HTTPS/TCP	Целевой порт контейнера для проверки.
Path	HTTP/HTTPS	Путь конечной точки (например, `/healthz`).
HTTP Headers	HTTP/HTTPS	Пользовательские заголовки (добавление пар ключ-значение).
Command	EXEC	Команда для проверки, выполняемая в контейнере (например, `sh -c "curl -I localhost:8080 | grep OK"`). **Примечание**: Экранируйте специальные символы и проверьте работоспособность команды.

Создание приложений из Chart

Создание на основе Helm Chart представляет собой нативный шаблон развертывания приложений. Helm Chart — это набор файлов, определяющих ресурсы Kubernetes, предназначенный для упаковки приложений и упрощения их распространения с возможностями управления версиями. Это обеспечивает беспрепятственный переход между средами, например, миграцию из среды разработки в продуктивную.

Содержание

Меры предосторожности

Если в кластере присутствуют как Linux, так и Windows узлы, необходимо обязательно настроить явный выбор узла, чтобы избежать конфликтов при планировании. Пример:

```
spec:
    spec:
      nodeSelector:
        kubernetes.io/os: linux
```

Предварительные требования

Если шаблон взят из приложения и ссылается на соответствующие ресурсы (например, секретные словари), убедитесь, что ресурсы, на которые будет ссылка, уже существуют в текущем namespace до развертывания приложения.

Процедура

1. В **Container Platform** перейдите в **Applications** > **Applications** в левой боковой панели.
2. Нажмите **Create**.
3. Выберите способ создания **Create from Catalog**.
4. Выберите Chart и настройте параметры, такие как `resources.requests`, `resources.limits` и другие параметры, тесно связанные с выбранным Chart.
5. Нажмите **Create**.

Веб-консоль перенаправит вас на страницу деталей **Application** > [**Native Applications**]. Процесс займет некоторое время, пожалуйста, подождите. В случае неудачи операции следуйте подсказкам интерфейса для завершения операции.

Справка по анализу статуса

Нажмите на ***Application Name*** для отображения подробного анализа статуса Chart в информации о деталях.

Тип	Причина
Initialized	Указывает статус загрузки шаблона Chart. • **True**: Шаблон Chart успешно загружен. • **False**: Загрузка шаблона Chart не удалась; конкретную причину ошибки можно посмотреть в столбце сообщения. • `ChartLoadFailed`: Не удалось загрузить шаблон Chart.

Тип	Причина
	• `InitializeFailed`: Произошло исключение в процессе инициализации до загрузки Chart.
Validated	Указывает статус проверки прав пользователя, зависимостей и других валидаций для шаблона Chart. • **True**: Все проверки прошли успешно. • **False**: Есть проверки, которые не прошли; конкретную причину ошибки можно посмотреть в столбце сообщения. • `DependenciesCheckFailed`: Проверка зависимостей Chart не пройдена. • `PermissionCheckFailed`: У текущего пользователя нет прав на выполнение операций с некоторыми ресурсами. • `ConsistentNamespaceCheckFailed`: При развертывании приложений через шаблоны в native applications Chart содержит ресурсы, требующие развертывания в разных namespace.
Synced	Указывает статус развертывания шаблона Chart. • **True**: Шаблон Chart успешно развернут. • **False**: Развертывание шаблона Chart не удалось; в столбце причины указано `ChartSyncFailed`, подробности ошибки можно посмотреть в столбце сообщения.

WARNING

- Если шаблон ссылается на ресурсы из других namespace, обратитесь к Администратору за помощью в создании. После этого вы сможете нормально обновлять и удалять Chart приложения через веб-консоль.
- Если шаблон ссылается на ресурсы уровня кластера (например, StorageClasses), рекомендуется обратиться к Администратору для помощи в создании.

Создание приложений из YAML

Если вы хорошо разбираетесь в синтаксисе YAML и предпочитаете определять конфигурации вне форм или предопределённых шаблонов, вы можете выбрать метод создания с помощью одного клика по YAML. Этот подход предоставляет более гибкую настройку базовой информации и ресурсов для вашего cloud-native приложения.

Содержание

Меры предосторожности

Если в кластере присутствуют как Linux, так и Windows узлы, чтобы предотвратить назначение приложения на несовместимые узлы, необходимо настроить выбор узлов. Например:

```
spec:
    spec:
      nodeSelector:
        kubernetes.io/os: linux
```

Предварительные условия

Убедитесь, что образы, определённые в YAML, могут быть загружены внутри текущего кластера. Вы можете проверить это с помощью команды `podman pull`.

Порядок действий

1. Перейдите в **Container Platform**, затем в **Application** > **Applications**.
2. Нажмите **Create**.
3. Выберите **Create from YAML**.
4. Заполните конфигурацию и нажмите **Create**.
5. Соответствующий **Deployment** можно просмотреть на странице деталей.

Создание приложений из кода

Создание приложения из кода реализовано с использованием технологии Source to Image (S2I). S2I — это автоматизированная платформа для построения контейнерных образов непосредственно из исходного кода. Такой подход стандартизирует и автоматизирует процесс сборки приложения, позволяя разработчикам сосредоточиться на разработке исходного кода без необходимости беспокоиться о деталях контейнеризации.

Содержание

Требования

- Завершите установку Alauda Container Platform Builds

Процедура

1. Перейдите в **Container Platform**, затем в **Application** > **Applications**.
2. Нажмите **Create**.
3. Выберите **Create from Code**.

4. Для подробного описания параметров обратитесь к разделу Managing applications created from Code
5. После заполнения параметров нажмите **Create**.
6. Соответствующий деплоймент можно просмотреть на странице **Detail Information**.

Creating applications from Operator Backed

Содержание

Понимание Operator Backed Application

Operator — это механизм расширения, построенный на основе Kubernetes Custom Controllers и Custom Resource Definitions (CRD), предназначенный для автоматизации полного жизненного цикла управления сложными приложениями. В рамках Alauda Container Platform, Operator Backed Application — это экземпляр приложения, созданный с помощью прединтегрированных или пользовательских Operators, при этом его рабочие процессы управляются Operator Lifecycle Manager (OLM). Это включает в себя ключевые процессы, такие как установка, обновления, разрешение зависимостей и контроль доступа.

Основные возможности

1. **Автоматизация сложных операций**: Operators преодолевают ограничения стандартных ресурсов Kubernetes (например, Deployment, StatefulSet) для решения задач управления stateful-приложениями, включая распределённую координацию, постоянное хранилище и версионированные поэтапные обновления. Пример: логика, закодированная в Operator, обеспечивает автономные операции для переключения отказоустойчивости кластера базы данных, согласованности данных между узлами и восстановления из резервных копий.
2. **Декларативная архитектура, управляемая состоянием**: Operators используют декларативные API на основе YAML для определения желаемого состояния приложения (например, spec.replicas: 5). Operators постоянно согласуют фактическое состояние с объявленным, обеспечивая возможности самовосстановления. Глубокая интеграция с GitOps-инструментами (например, Argo CD) гарантирует согласованность конфигураций среды.
3. **Интеллектуальное управление жизненным циклом**:
   - Поэтапные обновления и откат: объект Subscription в OLM подписывается на каналы обновлений (например, stable, alpha), инициируя автоматические итерации версий как для Operators, так и для управляемых ими приложений.
   - Разрешение зависимостей: Operators динамически определяют зависимости во время выполнения (например, конкретные драйверы хранения, CNI-плагины) для обеспечения успешного развертывания.
4. **Стандартизированная интеграция в экосистему**: OLM стандартизирует упаковку Operator (Bundle) и каналы распространения, позволяя развертывать производственные приложения (например, Etcd) из OperatorHub или приватных реестров одним кликом. Корпоративные расширения: Alauda Container Platform расширяет политики RBAC и возможности мультикластерного распространения для соответствия требованиям корпоративного уровня.

CRD Operator Backed Application

Этот Operator разработан и реализован с полным соблюдением стандартов и решений сообщества с открытым исходным кодом. Его Custom Resource Definition (CRD) продуманно включает лучшие практики и архитектурные шаблоны, широко применяемые в экосистеме Kubernetes. Референсные материалы по дизайну CRD:

1. CatalogSource ↗: Определяет источник пакетов Operator, доступных в кластере, таких как OperatorHub или пользовательские репозитории Operator.
2. ClusterServiceVersion (CSV) ↗: Основное определение метаданных для Operator, содержащее его имя, версию, предоставляемые API, необходимые разрешения, стратегию установки и подробную информацию по управлению жизненным циклом.
3. InstallPlan ↗: Фактический план выполнения установки Operator, автоматически создаваемый OLM на основе Subscription и CSV, описывающий конкретные шаги по созданию Operator и его зависимых ресурсов.
4. OperatorGroup ↗: Определяет набор целевых namespace, в которых Operator будет предоставлять свои сервисы и согласовывать ресурсы, а также ограничивает область действия разрешений RBAC Operator.
5. Subscription ↗: Используется для объявления конкретного Operator, который пользователь хочет установить и отслеживать в кластере, включая имя Operator, целевой канал (например, stable, alpha) и стратегию обновления. OLM использует Subscription для создания и управления установкой и обновлениями Operator.

Создание Operator Backed Application через веб-консоль

1. В **Container Platform** перейдите в **Applications** > **Applications** в левой боковой панели.
2. Нажмите **Create**.
3. Выберите способ создания **Create from Catalog**.
4. Выберите экземпляр Operator-Backed и настройте **Custom Resource Parameters**. Выберите экземпляр приложения, управляемого Operator, и настройте его спецификации Custom Resource (CR) в манифесте CR, включая:
   - `spec.resources.limits` (ограничения ресурсов на уровне контейнера).
   - `spec.resourceQuota` (политики квот, определённые Operator). Другие параметры CR, такие как `spec.replicas`, `spec.storage.className` и т.д.
5. Нажмите **Create**.

Веб-консоль перейдёт на страницу **Applications** > **Operator Backed Apps**.

INFO

Примечание: Процесс создания ресурсов в Kubernetes требует асинхронного согласования.

Завершение может занять несколько минут в зависимости от состояния кластера.

Устранение неполадок

Если создание ресурса не удалось:

1. Проверьте ошибки согласования контроллера:

```
kubectl get events --field-selector involvedObject.kind=<Your-Custom-Resource> --sort-by=.metadata.creationTimestamp
```

2. Проверьте доступность API ресурса:

```
kubectl api-resources | grep <Your-Resource-Type>
```

3. Повторите создание после проверки готовности CRD/Operator:

```
kubectl apply -f your-resource-manifest.yaml
```

Создание приложений с помощью CLI

`kubectl` — это основной интерфейс командной строки (CLI) для взаимодействия с кластерами Kubernetes. Он функционирует как клиент для Kubernetes API Server — RESTful HTTP API, который служит программным интерфейсом управляющей плоскости. Все операции Kubernetes доступны через API-эндпоинты, и `kubectl` по сути переводит команды CLI в соответствующие API-запросы для управления ресурсами кластера и рабочими нагрузками приложений (Deployments, StatefulSets и т.д.).

CLI-инструменты облегчают развертывание приложений, интеллектуально интерпретируя входные артефакты (образы, Chart и др.) и генерируя соответствующие объекты Kubernetes API. Создаваемые ресурсы зависят от типа входных данных:

- **Image**: напрямую создаёт Deployment.
- **Chart**: создаёт все объекты, определённые в Helm Chart.

Содержание

Предварительные требования

Плагин **Alauda Container Platform Web Terminal** установлен, а переключатель web-cli включён.

Процедура

1. В **Container Platform** нажмите на иконку терминала в правом нижнем углу.
2. Дождитесь инициализации сессии (1-3 секунды).
3. Выполните команды kubectl в интерактивной оболочке:

```
kubectl get pods -n ${CURRENT_NAMESPACE}
```

4. Просматривайте вывод команд в реальном времени

Пример

YAML

```yaml
# webapp.yaml
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: webapp
spec:
  componentKinds:
    - group: apps
      kind: Deployment
    - group: ""
      kind: Service
  descriptor: {}

# webapp-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  labels:
    app: webapp
    env: prod
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
        tier: frontend
    spec:
      containers:
      - name: webapp
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "250m"
```

```
            memory: "256Mi"
---
# webapp-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Команды kubectl

```
kubectl apply -f webapp.yaml -n {CURRENT_NAMESPACE}
kubectl apply -f webapp-deployment.yaml -n {CURRENT_NAMESPACE}
kubectl apply -f webapp-service.yaml -n {CURRENT_NAMESPACE}
```

Справка

- **Концептуальное руководство**: kubectl Overview ↗
- **Справочник по синтаксису**: kubectl Cheat Sheet ↗
- **Руководство по командам**: kubectl Commands ↗

Эксплуатация и сопровождение приложений

Развертывание приложений

Установка Alauda Container Pl

Предварительные требования

Установка Alauda Container Platform Arg

Обновление Alauda Container

Предварительные требования

Обновление Alauda Container Platform A

Application

Требования

Процедура

Application

Предварителы

Процедура

Описание статуса

Описание статуса

Приложения

Настройка HPA

Настройка HPA

Понимание Horizontal Pod Autoscalers

Предварительные требования

Создание горизонтального автоскейлера подов

Правила расчёта

Запуск и остановка приложений

Запуск и остановка приложений

Запуск приложения

Остановка приложения

Настройка VerticalPodAutoscaler (VPA)

Настройка VerticalPodAutoscaler (VPA)

Обзор

Понимание VerticalPodAutoscalers

Предварительные требования

Создание VerticalPodAutoscaler

Настройка CronHPA

Настройка CronHPA

Обновление приложений

Обновление приложений

Экспорт приложений

Экспорт приложений

Экспорт Helm Charts

Экспорт YAML локально

Экспорт YAML в репозиторий кода (Alpha)

Обновление и удаление Chart-приложений

Обновление и удаление Chart-приложений

Важные замечания

Предварительные требования

Описание анализа статуса

Управление версиями приложений

Управление версиями приложений

Создание снимка версии

Откат к исторической версии

Удаление приложений

Удаление приложений

Обработка ошибок нехватки ресурсов

Обработка ошибок нехватки ресурсов

Проверки состояния

Проверки состояния

Развертывание приложений

Установка Alauda Container Platform Argo Rollouts

Содержание

Предварительные требования

1. **Скачайте** установочный пакет плагина кластера **Alauda Container Platform Argo Rollouts**, соответствующий архитектуре вашей платформы.
2. **Загрузите** установочный пакет с помощью механизма Upload Packages.
3. **Установите** пакет в кластер 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. Для получения дополнительной информации обратитесь к CLI.

Установка Alauda Container Platform Argo Rollouts

Процедура

1. Перейдите в **Marketplace** > **Cluster Plugins**, чтобы открыть список **Cluster Plugins**.
2. Найдите плагин кластера **Alauda Container Platform Argo Rollouts**, нажмите **Install** и перейдите на страницу **Install Alauda Container Platform Argo Rollouts Plugin**.
3. Просто нажмите **Install**, чтобы завершить установку плагина кластера **Alauda Container Platform Argo Rollouts**.

INFO

Для версии v4.2.6 после установки необходимо выполнить следующие шаги:

1. Найдите экземпляр moduleinfo модуля **argo-rollouts** в кластере **global**:

   ```
   $ kubectl get moduleinfo | grep {cluster} | grep argo-rollouts
   ```

   Где `{cluster}` — имя рабочей нагрузки кластера.

2. На основе имени moduleinfo модуля **argo-rollouts** отредактируйте moduleinfo, добавив поле `valuesOverride`:

   ```
   $ kubectl edit moduleinfo {moduleinfo-name}
   ```

   Измените следующим образом:

```
spec:
  config: {}
  valuesOverride:
    acp/argoproj/chart/argo-rollouts:
      controller:
        initContainers:
        - args:
          - cp /bin/rollouts-plugin-trafficrouter-gatewayapi /plu
gins
          command:
          - /bin/sh
          - -c
          image: {registry address}/acp/argoproj-labs/rollouts-pl
ugin-trafficrouter-gatewayapi:v0.6.0-6
          name: copy-gwapi-plugin
          resources:
            requests:
              cpu: 100m
              memory: 64Mi
            limits:
              cpu: 200m
              memory: 128Mi
              ephemeral-storage: 500Mi
          volumeMounts:
            - mountPath: /plugins
              name: gwapi-plugin
```

Где `{registry address}` нужно заменить

Обновление Alauda Container Platform Argo Rollouts

Содержание

Предварительные требования

1. **Скачайте** установочный пакет плагина кластера **Alauda Container Platform Argo Rollouts**, соответствующий архитектуре вашей платформы.
2. **Загрузите** установочный пакет с помощью механизма Upload Packages.

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

Обновление Alauda Container Platform Argo Rollouts

Процедура

1. Войдите в Web Console и переключитесь в режим **Administrator**.
2. Перейдите в раздел **Clusters > Clusters**.
3. Выберите **workload cluster**, который хотите обновить, и откройте его страницу с деталями.
4. Перейдите на вкладку **Functional Components**.
5. Нажмите кнопку **Upgrade**.

INFO

Если вы обновляетесь до версии v4.2.6, необходимо выполнить следующие шаги:

1. Найдите экземпляр moduleinfo модуля **argo-rollouts** в кластере **global**:

   ```
   $ kubectl get moduleinfo | grep {cluster} | grep argo-rollouts
   ```

   Где `{cluster}` — имя workload кластера.

2. Исходя из имени moduleinfo модуля **argo-rollouts** вашего кластера, отредактируйте moduleinfo, добавив поле `valuesOverride`:

   ```
   $ kubectl edit moduleinfo {moduleinfo-name}
   ```

   Измените его следующим образом:

```
spec:
  config: {}
  valuesOverride:
    acp/argoproj/chart/argo-rollouts:
      controller:
        initContainers:
        - args:
          - cp /bin/rollouts-plugin-trafficrouter-gatewayapi /plu
gins
          command:
          - /bin/sh
          - -c
          image: {registry address}/acp/argoproj-labs/rollouts-pl
ugin-trafficrouter-gatewayapi:v0.6.0-6
          name: copy-gwapi-plugin
          resources:
            requests:
              cpu: 100m
              memory: 64Mi
            limits:
              cpu: 200m
              memory: 128Mi
              ephemeral-storage: 500Mi
          volumeMounts:
            - mountPath: /plugins
              name: gwapi-plugin
```

Где необходимо заменить `{registry address}`

Application Blue Green Deployment

В современном мире разработки программного обеспечения развертывание новых версий приложений является важной частью цикла разработки. Однако внедрение обновлений в производственные среды может быть рискованным процессом, так как даже небольшие проблемы могут привести к значительным простоям и потерям дохода. Развертывание по стратегии Blue-Green — это стратегия, которая снижает этот риск, обеспечивая возможность развертывания новых версий приложений без простоев.

Развертывание Blue-Green — это стратегия, при которой создаются две идентичные среды: «синяя» (blue) и «зелёная» (green). Синяя среда — это производственная среда, где в данный момент работает живая версия приложения, а зелёная — это непроизводственная среда, в которую развёртываются новые версии приложения.

Когда новая версия приложения готова к развертыванию, она разворачивается в зелёной среде. После того как новая версия развернута и протестирована, трафик переключается на зелёную среду, которая становится новой производственной средой. Синяя среда при этом становится непроизводственной, где можно развертывать будущие версии приложения.

Преимущества развертывания Blue Green

- Отсутствие простоев: Развертывания Blue-Green позволяют внедрять новые версии приложений без простоев, так как трафик плавно переключается с синей среды на зелёную.
- Лёгкий откат: Если новая версия приложения содержит ошибки, откат к предыдущей версии осуществляется легко, поскольку синяя среда всё ещё доступна.
- Снижение рисков: Использование стратегии Blue-Green значительно снижает риски при развертывании новых версий. Новая версия разворачивается и тестируется в

зелёной среде до переключения трафика с синей среды, что позволяет провести тщательное тестирование и уменьшить вероятность возникновения проблем в продакшене.

- Повышенная надёжность: Благодаря стратегии Blue-Green надёжность приложения повышается, так как синяя среда всегда доступна, и любые проблемы в зелёной среде можно быстро выявить и устранить без влияния на пользователей.
- Гибкость: Развертывания Blue-Green обеспечивают гибкость процесса развертывания. Несколько версий приложения могут работать параллельно, что облегчает тестирование и эксперименты.

Развертывание Blue Green с Argo Rollouts

Argo Rollouts — это контроллер Kubernetes и набор CRD, предоставляющие расширенные возможности развертывания, такие как blue-green, canary, анализ canary, эксперименты и прогрессивная доставка для Kubernetes.

Argo Rollouts (опционально) интегрируется с ingress-контроллерами и сервис-мешами, используя их возможности управления трафиком для постепенного переключения трафика на новую версию во время обновления. Кроме того, Rollouts может запрашивать и интерпретировать метрики от различных провайдеров для проверки ключевых KPI и автоматического продвижения или отката во время обновления.

С помощью Argo Rollouts вы можете автоматизировать развертывания blue green на кластерах Alauda Container Platform (ACP). Типичный процесс включает:

1. Определение ресурсов Rollout для управления разными версиями приложения.
2. Настройку сервисов Kubernetes для маршрутизации трафика между синей (текущей) и зелёной (новой) средами.
3. Развертывание новой версии в зелёной среде.
4. Проверку и тестирование новой версии.
5. Продвижение зелёной среды в продакшен путём переключения трафика.

Этот подход минимизирует простои и обеспечивает контролируемое, безопасное развертывание.

> **Ключевые понятия:**
>
> - **Rollout**: определение пользовательского ресурса (CRD) в Kubernetes, заменяющее стандартные ресурсы Deployment и позволяющее управлять развертываниями с расширенными возможностями, такими как blue-green и canary.

Содержание

Требования

1. ACP (Alauda Container Platform).
2. Kubernetes-кластер, управляемый ACP.
3. Установленный Argo Rollouts в кластере.
4. Плагин kubectl для Argo Rollouts.
5. Проект для создания в нём namespace.
6. Namespace в кластере, где будет развернуто приложение.

Процедура

1 Создание Deployment

Начните с определения «синей» версии вашего приложения. Это текущая версия, к которой будут обращаться пользователи. Создайте Kubernetes Deployment с нужным количеством реплик, версией образа контейнера (например, `hello:1.23.1`) и соответствующими метками, например `app=web`.

Используйте следующий YAML:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: hello:1.23.1
          ports:
            - containerPort: 80
```

Объяснение полей YAML:

- `apiVersion`: версия Kubernetes API, используемая для создания ресурса.
- `kind`: указывает, что это ресурс Deployment.
- `metadata.name`: имя Deployment.
- `spec.replicas`: желаемое количество реплик подов.

- `spec.selector.matchLabels`: определяет, какие поды управляются этим Deployment.
- `template.metadata.labels`: метки, применяемые к подам, используемые сервисами для выбора.
- `spec.containers`: контейнеры, запускаемые в каждом поде.
- `containers.name`: имя контейнера.
- `containers.image`: образ контейнера для запуска.
- `containers.ports.containerPort`: порт, открываемый контейнером.

Примените конфигурацию с помощью `kubectl`:

```
kubectl apply -f deployment.yaml
```

Это создаст производственную среду.

В качестве альтернативы можно использовать helm chart для создания deployments и сервисов.

2 Создание синего сервиса

Создайте Kubernetes `Service`, который будет открывать доступ к синему Deployment. Этот сервис будет перенаправлять трафик на синие поды на основе совпадающих меток. Изначально селектор сервиса нацелен на поды с меткой `app=web`.

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Объяснение полей YAML:

- `apiVersion`: версия Kubernetes API для создания сервиса.
- `kind`: указывает, что это ресурс Service.
- `metadata.name`: имя сервиса.
- `spec.selector`: определяет поды, на которые будет направлен трафик, по меткам.
- `ports.protocol`: используемый протокол (TCP).
- `ports.port`: порт, открываемый сервисом.
- `ports.targetPort`: порт контейнера, на который направляется трафик.

Примените с помощью:

```
kubectl apply -f web-service.yaml
```

Это обеспечит внешний доступ к синему Deployment.

3 Проверка синего Deployment

Убедитесь, что синий Deployment работает корректно, перечислив поды:

```
kubectl get pods -l app=web
```

Проверьте, что все ожидаемые реплики (2) находятся в состоянии `Running`. Это гарантирует, что приложение готово обслуживать трафик.

4 Проверка маршрутизации трафика на синий Deployment

Убедитесь, что сервис `web` корректно перенаправляет трафик на синие поды. Используйте команду:

```
kubectl describe service web | grep Endpoints
```

В выводе должны быть IP-адреса синих подов — это конечные точки, получающие трафик.

5 Создание Rollout

Далее создайте ресурс `Rollout` из Argo Rollouts со стратегией `BlueGreen`.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-bluegreen
spec:
  replicas: 2
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: web
  workloadRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web
    scaleDown: onsuccess
  strategy:
    blueGreen:
      activeService: web
      autoPromotionEnabled: false
```

Объяснение полей YAML:

- `spec.selector`: селектор меток для подов. Существующие ReplicaSet, поды которых соответствуют этому селектору, будут затронуты этим rollout. Должен совпадать с метками шаблона пода.
- `workloadRef`: ссылка на workload и стратегия масштабирования после миграции на Rollout.
  - `scaleDown`: указывает, будет ли workload (Deployment) масштабироваться вниз после успешного Rollout. Возможные значения:
    - "never": Deployment не масштабируется вниз.

    - "onsuccess": Deployment масштабируется вниз после того, как Rollout становится здоровым.
    - "progressively": по мере масштабирования Rollout Deployment масштабируется вниз. Если Rollout неудачен, Deployment масштабируется обратно.
- `strategy`: стратегия развертывания, поддерживает `BlueGreen` и `Canary`.
    - `blueGreen`: определение стратегии BlueGreen.
        - `activeService`: сервис, который обновляется с новым хешем шаблона при продвижении. Обязательное поле для стратегии blueGreen.
        - `autoPromotionEnabled`: если отключено, автоматическое продвижение новой версии приостанавливается сразу перед переключением. По умолчанию новая версия продвигается, как только ReplicaSet полностью готов. Продвижение можно возобновить командой: `kubectl argo rollouts promote ROLLOUT`

Примените с помощью:

```
kubectl apply -f rollout.yaml
```

Это настроит rollout для развертывания со стратегией `BlueGreen`.

6 Проверка Rollout

После создания `Rollout` Argo Rollouts создаст новый ReplicaSet с тем же шаблоном, что и у Deployment. Пока поды нового ReplicaSet здоровы, Deployment масштабируется до 0.

Используйте команду для проверки состояния подов:

```
kubectl argo rollouts get rollout rollout-bluegreen
Name:            rollout-bluegreen
Namespace:       default
Status:          ✔ Healthy
Strategy:        BlueGreen
Images:          hello:1.23.1 (stable, active)
Replicas:
  Desired:       2
  Current:       2
  Updated:       2
  Ready:         2
  Available:     2

NAME                                       KIND        STATUS
AGE  INFO
⟳ rollout-bluegreen                         Rollout     ✔ Health
y  95s
└──# revision:1
   └──⧉ rollout-bluegreen-595d4567cc        ReplicaSet  ✔ Healthy
18s  stable,active
      ├──□ rollout-bluegreen-595d4567cc-mc769  Pod         ✔ Running
8s   ready:1/1
      └──□ rollout-bluegreen-595d4567cc-zdc5x  Pod         ✔ Running
8s   ready:1/1
```

Сервис `web` будет перенаправлять трафик на поды, созданные rollout. Проверьте это командой:

```
kubectl describe service web | grep Endpoints
```

7 Подготовка зелёного Deployment

Далее подготовьте новую версию приложения как зелёный Deployment. Обновите Deployment `web`, указав новую версию образа (например, `hello:1.23.2`).

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: hello:1.23.2
          ports:
            - containerPort: 80
```

Объяснение полей YAML:

- Идентично исходному Deployment, за исключением:
  - `containers.image`: обновлено на новую версию образа.

Примените с помощью:

```
kubectl apply -f deployment.yaml
```

Это подготовит новую версию приложения для тестирования.

Rollout создаст новый ReplicaSet для управления зелёными подами, а трафик по-прежнему будет направляться на синие поды. Проверьте состояние командой:

```
kubectl argo rollouts get rollout rollout-bluegreen
Name:            rollout-bluegreen
Namespace:       default
Status:          ॥ Paused
Message:         BlueGreenPause
Strategy:        BlueGreen
Images:          hello:1.23.1 (stable, active)
                 hello:1.23.2
Replicas:
  Desired:       2
  Current:       4
  Updated:       2
  Ready:         2
  Available:     2

NAME                                       KIND        STATUS
AGE  INFO
⟳ rollout-bluegreen                        Rollout     ॥ Paused
14m
├──# revision:2
│  └──⧉ rollout-bluegreen-776b688d57       ReplicaSet  ✔ Healthy
24s
│     ├──□ rollout-bluegreen-776b688d57-kxr66  Pod      ✔ Running
23s  ready:1/1
│     └──□ rollout-bluegreen-776b688d57-vv7t7  Pod      ✔ Running
23s  ready:1/1
└──# revision:1
   └──⧉ rollout-bluegreen-595d4567cc       ReplicaSet  ✔ Healthy
12m  stable,active
      ├──□ rollout-bluegreen-595d4567cc-mc769  Pod      ✔ Running
12m  ready:1/1
      └──□ rollout-bluegreen-595d4567cc-zdc5x  Pod      ✔ Running
12m  ready:1/1
```

В данный момент запущено 4 пода — синие и зелёные версии. Активный сервис направлен на синюю версию, процесс rollout приостановлен.

Если вы используете helm chart для развертывания приложения, используйте helm для обновления приложения до зелёной версии.

8 Продвижение Rollout на зелёную версию

Когда зелёная версия готова, продвиньте rollout, чтобы переключить трафик на зелёные поды. Используйте команду:

```
kubectl argo rollouts promote rollout-bluegreen
```

Для проверки завершения rollout:

```
kubectl argo rollouts get rollout rollout-bluegreen
Name:            rollout-bluegreen
Namespace:       default
Status:          ✔ Healthy
Strategy:        BlueGreen
Images:          hello:1.23.2 (stable, active)
Replicas:
  Desired:       2
  Current:       2
  Updated:       2
  Ready:         2
  Available:     2

NAME                                       KIND        STATUS
AGE   INFO
⟳ rollout-bluegreen                         Rollout     ✔ Health
y      3h2m
├──# revision:2
│  └──⧉ rollout-bluegreen-776b688d57        ReplicaSet  ✔ Healthy
168m  stable,active
│     ├──□ rollout-bluegreen-776b688d57-kxr66  Pod       ✔ Running
168m  ready:1/1
│     └──□ rollout-bluegreen-776b688d57-vv7t7  Pod       ✔ Running
168m  ready:1/1
└──# revision:1
   └──⧉ rollout-bluegreen-595d4567cc        ReplicaSet  • ScaledD
own   3h1m
      ├──□ rollout-bluegreen-595d4567cc-mc769  Pod       ◌ Terminat
ing  3h    ready:1/1
      └──□ rollout-bluegreen-595d4567cc-zdc5x  Pod       ◌ Terminat
ing  3h    ready:1/1
```

> Если активный образ (`Images`) обновлён на `hello:1.23.2`, а синий ReplicaSet масштабирован до 0, значит rollout завершён.

Application Canary Deployment

Канареечное развертывание — это стратегия постепенного выпуска, при которой новая версия приложения постепенно вводится для небольшой части пользователей или трафика. Такой поэтапный выпуск позволяет командам отслеживать поведение системы, собирать метрики и обеспечивать стабильность перед полномасштабным развертыванием. Этот подход значительно снижает риски, особенно в продуктивных средах.

Argo Rollouts — это контроллер прогрессивного развертывания, нативный для Kubernetes, который облегчает использование продвинутых стратегий развертывания. Он расширяет возможности Kubernetes, предлагая такие функции, как Canary, Blue-Green Deployments, Analysis Runs, Experimentation и автоматические откаты. Интегрируется со стеком наблюдаемости для проверки состояния на основе метрик и предоставляет управление доставкой приложений через CLI и панель управления.

> **Ключевые понятия:**
>
> - **Rollout**: определение пользовательского ресурса (CRD) в Kubernetes, заменяющее стандартные ресурсы Deployment и позволяющее использовать продвинутый контроль развертывания, например blue-green или canary.
> - **Canary Steps**: серия поэтапных действий по перенаправлению трафика, например сначала 25%, затем 50% трафика на новую версию.
> - **Pause Steps**: вводят интервалы ожидания для ручной или автоматической проверки перед переходом к следующему шагу канареечного развертывания.

Преимущества канареечных развертываний

- **Снижение рисков**: развертывая изменения сначала на небольшой части серверов, можно выявить проблемы и устранить их до полного выпуска, минимизируя влияние на пользователей.
- **Пошаговый выпуск**: такой подход позволяет постепенно вводить новые функции, что помогает эффективно отслеживать производительность и отзывы пользователей.
- **Обратная связь в реальном времени**: канареечные развертывания дают мгновенное представление о производительности и стабильности новых релизов в реальных условиях.
- **Гибкость**: можно корректировать процесс развертывания на основе метрик производительности. Это позволяет динамично приостанавливать или откатывать выпуск при необходимости.
- **Экономия ресурсов**: в отличие от blue/green развертываний, канареечные не требуют отдельной среды, что делает их более ресурсосберегающими.

Канареечные развертывания с Argo Rollouts

Argo Rollouts поддерживает стратегию канареечного развертывания для выпуска и управления трафиком через Gateway API Plugin. В ACP можно использовать ALB в качестве Gateway API Provider для реализации контроля трафика для Argo Rollouts.

Содержание

Предварительные требования

1. Argo Rollouts с установленным Gateway API плагином в кластере.
2. kubectl плагин Argo Rollouts (установка доступна здесь ↗).
3. Проект, в котором будет создан namespace.
4. ALB, развернутый в кластере и выделенный проекту.
5. Namespace в кластере, куда будет развернуто приложение.

Процедура

1 Создание Deployment

Начните с определения «стабильной» версии вашего приложения. Это текущая версия, к которой будут обращаться пользователи. Создайте Kubernetes Deployment с нужным количеством реплик, версией образа контейнера (например, `hello:1.23.1`) и соответствующими метками, например `app=web`.

Используйте следующий YAML:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: hello:1.23.1
          ports:
            - containerPort: 80
```

Объяснение полей YAML:

- `apiVersion`: версия Kubernetes API, используемая для создания ресурса.
- `kind`: указывает, что это ресурс Deployment.
- `metadata.name`: имя Deployment.
- `spec.replicas`: желаемое количество реплик подов.
- `spec.selector.matchLabels`: определяет, какие поды управляются Deployment.
- `template.metadata.labels`: метки, применяемые к подам, используемые сервисами для выбора.
- `spec.containers`: контейнеры, запускаемые в каждом поде.
- `containers.name`: имя контейнера.
- `containers.image`: образ контейнера для запуска.
- `containers.ports.containerPort`: порт, открываемый контейнером.

Примените конфигурацию с помощью `kubectl`:

```
kubectl apply -f deployment.yaml
```

Это создаст рабочее окружение.

В качестве альтернативы можно использовать helm chart для создания deployments и services.

2 Создание стабильного сервиса

Создайте Kubernetes `Service`, который будет экспонировать стабильный Deployment. Этот сервис будет перенаправлять трафик на поды стабильной версии на основе совпадающих меток. Изначально селектор сервиса нацелен на поды с меткой `app=web`.

```
apiVersion: v1
kind: Service
metadata:
  name: web-stable
spec:
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Объяснение полей YAML:

- `apiVersion`: версия Kubernetes API для создания Service.
- `kind`: указывает, что это ресурс Service.
- `metadata.name`: имя сервиса.
- `spec.selector`: определяет поды, на которые будет направлен трафик, по меткам.
- `ports.protocol`: используемый протокол (TCP).
- `ports.port`: порт, открываемый сервисом.
- `ports.targetPort`: порт контейнера, на который направляется трафик.

Примените с помощью:

```
kubectl apply -f web-stable-service.yaml
```

Это обеспечит внешний доступ к стабильному развертыванию.

3 Создание канареечного сервиса

Создайте Kubernetes `Service`, который будет экспонировать канареечный Deployment. Этот сервис будет перенаправлять трафик на поды канареечной версии на основе совпадающих меток. Изначально селектор сервиса нацелен на поды с меткой `app=web`.

```
apiVersion: v1
kind: Service
metadata:
  name: web-canary
spec:
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Объяснение полей YAML:

- `apiVersion`: версия Kubernetes API для создания Service.
- `kind`: указывает, что это ресурс Service.
- `metadata.name`: имя сервиса.
- `spec.selector`: определяет поды, на которые будет направлен трафик, по меткам.
- `ports.protocol`: используемый протокол (TCP).
- `ports.port`: порт, открываемый сервисом.
- `ports.targetPort`: порт контейнера, на который направляется трафик.

Примените с помощью:

```
kubectl apply -f web-canary-service.yaml
```

Это обеспечит внешний доступ к канареечному развертыванию.

4 Создание Gateway

Используя домен `example.com` для доступа к сервису, создайте gateway для экспонирования сервиса с этим доменом:

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: default
spec:
  gatewayClassName: exclusive-gateway
  listeners:
  - allowedRoutes:
      namespaces:
        from: All
    name: gateway-metric
    port: 11782
    protocol: TCP
  - allowedRoutes:
      namespaces:
        from: All
    hostname: example.com
    name: web
    port: 80
    protocol: HTTP
```

Выполните команду:

```
kubectl apply -f gateway.yaml
```

Gateway получит внешний IP-адрес, его можно узнать из поля `status.addresses` типа `IPAddress` в ресурсе gateway.

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: default
...
status:
  addresses:
  - type: IPAddress
    value: 192.168.134.30
```

5 Настройка DNS

Настройте ваш DNS-сервер так, чтобы домен разрешался в IP-адрес gateway.

Проверьте разрешение DNS командой:

```
nslookup example.com
Server:         192.168.16.19
Address:        192.168.16.19#53

Non-authoritative answer:
Name:   example.com
Address: 192.168.134.30
```

В ответе должен быть IP-адрес gateway.

6 Создание HTTPRoute

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: web
spec:
  hostnames:
  - example.com
  parentRefs:
  - group: gateway.networking.k8s.io
    kind: Gateway
    name: default
    namespace: default
    sectionName: web
  rules:
  - backendRefs:
    - group: ""
      kind: Service
      name: web-canary
      namespace: default
      port: 80
      weight: 0
    - group: ""
      kind: Service
      name: web-stable
      namespace: default
      port: 80
      weight: 100
    matches:
    - path:
        type: PathPrefix
        value: /
```

Примените с помощью:

```bash
kubectl apply -f httproute.yaml
```

7 Доступ к стабильному сервису

Вне кластера используйте команду для доступа к сервису по домену:

```
curl http://example.com
```

Или откройте `http://example.com` в браузере.

8 Создание Rollout

Далее создайте ресурс `Rollout` из Argo Rollouts со стратегией `Canary`.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-canary
spec:
  minReadySeconds: 30
  replicas: 2
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: web
  strategy:
    canary:
      canaryService: web-canary
      maxSurge: 25%
      maxUnavailable: 0
      stableService: web-stable
      steps:
      - setWeight: 50
      - pause: {}
      - setWeight: 100
      trafficRouting:
        plugins:
          argoproj-labs/gatewayAPI:
            httpRoute: web
            namespace: default
  workloadRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web
    scaleDown: onsuccess
```

Объяснение полей YAML:

- `spec.selector`: селектор меток для подов. Существующие ReplicaSets, поды которых соответствуют этому селектору, будут затронуты этим rollout. Должен совпадать с метками шаблона пода.
- `workloadRef`: ссылка на workload и стратегия масштабирования для применения rollout.
- `scaleDown`: указывает, будет ли workload (Deployment) масштабироваться вниз после миграции на Rollout. Возможные значения:
  - "never": Deployment не масштабируется вниз.
  - "onsuccess": Deployment масштабируется вниз после успешного завершения Rollout.
  - "progressively": при масштабировании Rollout вверх Deployment масштабируется вниз. Если Rollout неудачен, Deployment масштабируется обратно вверх.
- `strategy`: стратегия rollout, поддерживает `BlueGreen` и `Canary`.
- `canary`: определение стратегии канареечного rollout.
  - `canaryService`: ссылка на сервис, который контроллер будет обновлять для выбора канареечных подов. Обязательно для маршрутизации трафика.
  - `stableService`: ссылка на сервис, который контроллер будет обновлять для выбора стабильных подов. Обязательно для маршрутизации трафика.
  - `steps`: последовательность шагов обновления канареечного релиза. Пропускается при первоначальном развертывании rollout.
    - `setWeight`: устанавливает долю трафика для канареечного ReplicaSet.
    - `pause`: приостанавливает rollout на неопределённое время или на заданный период. Поддерживаемые единицы: s, m, h. `{}` означает неопределённо.
    - `plugin`: выполняет настроенный плагин, здесь используется плагин `gatewayAPI`.

Примените с помощью:

```
kubectl apply -f rollout.yaml
```

Это настроит rollout с стратегией `Canary`. Изначально вес будет установлен в 50, и rollout приостановится для проверки. 50% трафика будет направлено на канареечный сервис. После подтверждения rollout вес будет увеличен до 100, и весь трафик пойдёт на канареечный сервис. В итоге канареечный сервис станет стабильным.

9 Проверка Rollout

После создания `Rollout` Argo Rollouts создаст новый ReplicaSet с тем же шаблоном, что и у Deployment. Пока поды нового ReplicaSet здоровы, Deployment масштабируется до 0.

Используйте команду для проверки состояния подов:

```
kubectl argo rollouts get rollout rollout-canary
Name:            rollout-canary
Namespace:       default
Status:          ✔ Healthy
Strategy:        Canary
Step:            9/9
SetWeight:       100
ActualWeight:    100
Images:          hello:1.23.1 (stable)
Replicas:
Desired:         2
Current:         2
Updated:         2
Ready:           2
Available:       2

NAME                                       KIND        STATUS     AGE
INFO
⟳ rollout-canary                             Rollout     ✔ Healthy
32s
└──# revision:1
   └──⧉ rollout-canary-5c9d79697b            ReplicaSet  ✔ Healthy  32
s  stable
     ├──□ rollout-canary-5c9d79697b-fh78d    Pod         ✔ Running  32s
ready:1/1
     └──□ rollout-canary-5c9d79697b-rrbtj    Pod         ✔ Running  32s
ready:1/1
```

10 Подготовка канареечного развертывания

Далее подготовьте новую версию приложения как зеленое развертывание. Обновите Deployment `web` с новой версией образа (например, `hello:1.23.2`). Используйте команду:

```
kubectl patch deployment web -p '{"spec":{"template":{"spec":{"contai
ners":[{"name":"web","image":"hello:1.23.2"}]}}}}'
```

Это подготовит новую версию приложения для тестирования.

Rollout создаст новый ReplicaSet для управления канареечными подами, и 50% трафика будет направлено на канареечные поды. Проверьте состояние командой:

```
kubectl argo rollouts get rollout rollout-canary
Name:            rollout-canary
Namespace:       default
Status:          ॥ Paused
Message:         CanaryPauseStep
Strategy:        Canary
Step:            1/3
SetWeight:       50
ActualWeight:    50
Images:          hello:1.23.1 (stable)
                 hello:1.23.2 (canary)
Replicas:
Desired:         2
Current:         3
Updated:         1
Ready:           3
Available:       3

NAME                                       KIND        STATUS     AGE
INFO
⟳ rollout-canary                             Rollout     ॥ Paused   9
5s
├──# revision:2
│  └──⧉ rollout-canary-5898765588            ReplicaSet  ✔ Healthy  4
6s  canary
│     └──□ rollout-canary-5898765588-ls5jk   Pod         ✔ Running  4
5s  ready:1/1
└──# revision:1
   └──⧉ rollout-canary-5c9d79697b            ReplicaSet  ✔ Healthy  95
s  stable
    ├──□ rollout-canary-5c9d79697b-fk269     Pod         ✔ Running  94s
ready:1/1
    └──□ rollout-canary-5c9d79697b-wkmcn     Pod         ✔ Running  94s
ready:1/1
```

В данный момент запущено 3 пода с версиями stable и canary. Вес установлен в 50, 50% трафика направляется на канареечный сервис. Процесс rollout приостановлен в ожидании подтверждения.

Если вы используете helm chart для развертывания приложения, используйте helm для обновления приложения до канареечной версии.

При обращении к `http://example.com` 50% трафика будет направлено на канареечный сервис. Вы должны увидеть различия в ответах от URL.

11

Продвижение Rollout

Когда канареечная версия протестирована и работает корректно, вы можете продвинуть rollout, переключив весь трафик на канареечные поды. Используйте команду:

```
kubectl argo rollouts promote rollout-canary
```

Для проверки завершения rollout:

```
kubectl argo rollouts get rollout rollout-canary
Name:            rollout-canary
Namespace:       default
Status:          ✔ Healthy
Strategy:        Canary
Step:            3/3
SetWeight:       100
ActualWeight:    100
Images:          hello:1.23.2 (stable)
Replicas:
Desired:         2
Current:         2
Updated:         2
Ready:           2
Available:       2

NAME                                       KIND        STATUS
AGE    INFO
⟳ rollout-canary                             Rollout     ✔ Healthy
8m42s
├──# revision:2
│  └──⧉ rollout-canary-5898765588            ReplicaSet  ✔ Healthy
7m53s  stable
│     ├──□ rollout-canary-5898765588-ls5jk   Pod         ✔ Running
7m52s  ready:1/1
│     └──□ rollout-canary-5898765588-dkfwg   Pod         ✔ Running
68s    ready:1/1
└──# revision:1
   └──⧉ rollout-canary-5c9d79697b            ReplicaSet  • ScaledDown
8m42s
    ├──□ rollout-canary-5c9d79697b-fk269     Pod         ◌ Terminating
8m41s  ready:1/1
    └──□ rollout-canary-5c9d79697b-wkmcn     Pod         ◌ Terminating
8m41s  ready:1/1
```

Если в поле `Images` стабильной версии обновлено на `hello:1.23.2`, а ReplicaSet ревизии 1 масштабирован до 0, значит rollout завершён.

При обращении к `http://example.com` 100% трафика будет направлено на канареечный сервис.

12

Прерывание Rollout (опционально)

Если во время rollout вы обнаружили проблемы с канареечной версией, вы можете прервать процесс и переключить весь трафик обратно на стабильный сервис. Используйте команду:

```
kubectl argo rollouts abort rollout-canary
```

Для проверки результата:

```
kubectl argo rollouts get rollout rollout-canary
Name:            rollout-demo
Namespace:       default
Status:          ✖ Degraded
Message:         RolloutAborted: Rollout aborted update to revision 3
Strategy:        Canary
Step:          0/3
SetWeight:     0
ActualWeight:  0
Images:          hello:1.23.1 (stable)
Replicas:
Desired:       2
Current:       2
Updated:       0
Ready:         2
Available:     2

NAME                                       KIND        STATUS        A
GE  INFO
⟳ rollout-canary                             Rollout     ✖ Degraded
18m
├──# revision:3
│  └──⧉ rollout-canary-5c9d79697b            ReplicaSet  • ScaledDown
18m  canary,delay:passed
└──# revision:2
   └──⧉ rollout-canary-5898765588            ReplicaSet  ✔ Healthy
17m  stable
     ├──□ rollout-canary-5898765588-ls5jk    Pod         ✔ Running
17m  ready:1/1
     └──□ rollout-canary-5898765588-dkfwg    Pod         ✔ Running
10m  ready:1/1
```

При обращении к `http://example.com` 100% трафика будет направлено на стабильный сервис.

Описание статуса

Содержание

Приложения

Статусы нативных приложений и их соответствующие значения приведены ниже. Числа после статуса указывают количество вычислительных компонентов.

Статус	Значение
Running	Все вычислительные компоненты работают в нормальном режиме.
Partially Running	Некоторые вычислительные компоненты работают, а другие остановлены.
Stopped	Все вычислительные компоненты остановлены.
Processing	По крайней мере один вычислительный компонент находится в состоянии ожидания.
No Computing Components	В приложении отсутствуют вычислительные компоненты.
Failed	Развертывание не удалось.

Примечание: Аналогично, числа в статусе вычислительного компонента указывают количество групп контейнеров.

Развертывание

- Running: Все Pod работают в нормальном режиме.
- Processing: Есть Pod, которые не находятся в состоянии Running.
- Stopped: Все Pod остановлены.
- Failed: Развертывание не удалось.

Настройка HPA

HPA (Horizontal Pod Autoscaler) автоматически масштабирует количество подов вверх или вниз на основе заданных политик и метрик, позволяя приложениям справляться с внезапными всплесками нагрузки при оптимальном использовании ресурсов в периоды низкой активности.

Содержание

Понимание Horizontal Pod Autoscalers

Вы можете создать горизонтальный автоскейлер подов, чтобы указать минимальное и максимальное количество подов, которые вы хотите запустить, а также целевое использование CPU или памяти для ваших подов.

После создания горизонтального автоскейлера платформа начинает опрашивать метрики ресурсов CPU и/или памяти на подах. Когда эти метрики становятся доступны, автоскейлер вычисляет отношение текущего использования метрики к желаемому значению и масштабирует количество подов вверх или вниз соответственно. Запрос метрик и масштабирование происходят с регулярным интервалом, но может пройти от одной до двух минут, прежде чем метрики станут доступны.

Для replication controllers такое масштабирование напрямую соответствует количеству реплик replication controller. Для deployment configurations масштабирование напрямую соответствует количеству реплик deployment configuration. Обратите внимание, что автоскейлинг применяется только к последнему деплою в фазе Complete.

Платформа автоматически учитывает ресурсы и предотвращает ненужное автоскейлинг во время всплесков ресурсов, например, при запуске. Поды в состоянии unready имеют 0 использования CPU при масштабировании вверх, а автоскейлер игнорирует такие поды при масштабировании вниз. Поды без известных метрик считаются с 0% использования CPU при масштабировании вверх и 100% CPU при масштабировании вниз. Это обеспечивает большую стабильность при принятии решений HPA. Для использования этой функции необходимо настроить readiness checks, чтобы определить, готов ли новый под к использованию.

Как работает HPA?

Горизонтальный автоскейлер подов (HPA) расширяет концепцию автоскейлинга подов. HPA позволяет создавать и управлять группой балансируемых по нагрузке узлов. HPA автоматически увеличивает или уменьшает количество подов при превышении заданного порога CPU или памяти.

HPA работает как управляющий цикл с периодом синхронизации по умолчанию 15 секунд. В этот период controller manager опрашивает использование CPU, памяти или обоих параметров в соответствии с конфигурацией HPA. Controller manager получает

метрики использования из resource metrics API для каждого пода, на который направлен HPA.

Если задана целевая величина использования, контроллер вычисляет значение использования в процентах от соответствующего запроса ресурса на контейнерах каждого пода. Затем контроллер берет среднее значение использования по всем целевым подам и формирует коэффициент, который используется для масштабирования желаемого количества реплик.

Поддерживаемые метрики

Горизонтальные автоскейлеры подов поддерживают следующие метрики:

Метрика	Описание
CPU Utilization	Количество используемых ядер CPU. Может использоваться для расчёта процента от запрошенного CPU пода.
Memory Utilization	Объем используемой памяти. Может использоваться для расчёта процента от запрошенной памяти пода.
Network Inbound Traffic	Объем входящего сетевого трафика в под, измеряется в KiB/s.
Network Outbound Traffic	Объем исходящего сетевого трафика из пода, измеряется в KiB/s.
Storage Read Traffic	Объем данных, читаемых из хранилища, измеряется в KiB/s.
Storage Write Traffic	Объем данных, записываемых в хранилище, измеряется в KiB/s.

> **Важно**: Для автоскейлинга на основе памяти использование памяти должно пропорционально увеличиваться и уменьшаться вместе с количеством реплик. В среднем:
>
> - Увеличение количества реплик должно приводить к общему снижению использования памяти (working set) на под.

> - Уменьшение количества реплик должно приводить к общему увеличению использования памяти на под.
> - Используйте платформу для проверки поведения памяти вашего приложения и убедитесь, что оно соответствует этим требованиям перед использованием автоскейлинга на основе памяти.

Предварительные требования

Убедитесь, что компоненты мониторинга развернуты в текущем кластере и работают корректно. Вы можете проверить статус развертывания и состояние компонентов мониторинга, нажав в правом верхнем углу платформы ? > **Platform Health Status**..

Создание горизонтального автоскейлера подов

Использование CLI

Вы можете создать горизонтальный автоскейлер подов с помощью командной строки, определив YAML-файл и используя команду `kubectl create`. В следующем примере показан автоскейлинг для объекта Deployment. Изначально развертывание требует 3 пода. Объект HPA увеличивает минимум до 5. Если использование CPU на подах достигает 75%, количество подов увеличивается до 7:

1. Создайте YAML-файл с именем `hpa.yaml` со следующим содержимым:

```yaml
apiVersion: autoscaling/v2 (1)
kind: HorizontalPodAutoscaler (2)
metadata:
  name: hpa-demo (3)
  namespace: default
spec:
  maxReplicas: 7 (4)
  minReplicas: 3 (5)
  scaleTargetRef:
    apiVersion: apps/v1 (6)
    kind: Deployment (7)
    name: deployment-demo (8)
  targetCPUUtilizationPercentage: 75 (9)
```

(1) Используйте API autoscaling/v2.

(2) Имя ресурса HPA.

(3) Имя деплоя для масштабирования.

(4) Максимальное количество реплик для масштабирования вверх.

(5) Минимальное количество реплик для поддержания.

(6) Укажите версию API объекта для масштабирования.

(7) Укажите тип объекта. Объект должен быть Deployment, ReplicaSet или StatefulSet.

(8) Целевой ресурс, к которому применяется HPA.

(9) Целевой процент использования CPU, при котором происходит масштабирование.

2. Примените YAML-файл для создания HPA:

```
kubectl create -f hpa.yaml
```

Пример вывода:

```
horizontalpodautoscaler.autoscaling/hpa-demo created
```

3. После создания HPA вы можете просмотреть текущее состояние деплоя, выполнив команду:

```
kubectl get deployment deployment-demo
```

Пример вывода:

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deployment-demo   5/5     5            5           3m
```

4. Также можно проверить статус вашего HPA:

```
kubectl get hpa hpa-demo
```

Пример вывода:

```
NAME         REFERENCE                     TARGETS    MINPODS   MAXPODS
REPLICAS   AGE
hpa-demo   Deployment/deployment-demo   0%/75%     3         7
3          2m
```

Использование веб-консоли

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Workloads** > **Deployments**.
3. Кликните на ***Имя Deployment***.
4. Прокрутите вниз до раздела **Elastic Scaling** и нажмите **Update** справа.
5. Выберите **Horizontal Scaling** и заполните конфигурацию политики.

Параметр	Описание
Pod Count	После успешного создания деплоя необходимо оценить **Минимальное количество подов**, соответствующее известным и регулярным изменениям бизнес-объема, а также **Максимальное количество подов**, которое может поддерживаться квотой namespace при высокой нагрузке. Максимальное или минимальное количество подов можно изменять после настройки, рекомендуется сначала получить более точное значение через нагрузочное тестирование и затем

Параметр	Описание
	корректировать в процессе эксплуатации для удовлетворения бизнес-потребностей.
Trigger Policy	Перечислите **Метрики**, чувствительные к изменениям бизнеса, и их **Целевые пороги**, при достижении которых происходит масштабирование вверх или вниз. Например, если вы установите *CPU Utilization = 60%*, то при отклонении использования CPU от 60% платформа начнет автоматически корректировать количество подов в зависимости от недостатка или избытка ресурсов текущего деплоя. **Примечание**: Типы метрик включают встроенные и пользовательские. Пользовательские метрики применимы только к деплоям в нативных приложениях, и их необходимо предварительно добавить custom metrics .
Scale Up/Down Step (Alpha)	Для бизнесов с особыми требованиями к скорости масштабирования можно постепенно адаптироваться к изменениям объема, задавая **Шаг масштабирования вверх** или **Шаг масштабирования вниз**. Для шага масштабирования вниз можно настроить **Окно стабильности**, по умолчанию 300 секунд, что означает необходимость ожидания 300 секунд перед выполнением действия масштабирования вниз.

6. Нажмите **Update**.

Использование пользовательских метрик для HPA

HPA с пользовательскими метриками расширяет оригинальный HorizontalPodAutoscaler, поддерживая дополнительные метрики помимо CPU и памяти.

Требования

- kube-controller-manager: horizontal-pod-autoscaler-use-rest-clients=true
- Предустановленный metrics-server
- Prometheus

- custom-metrics-api

Традиционный (Core Metrics) HPA

Традиционный HPA поддерживает метрики использования CPU и памяти для динамического изменения количества экземпляров Pod, как показано в примере ниже:

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-app-nginx
  namespace: test-namespace
spec:
  maxReplicas: 1
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-app-nginx
  targetCPUUtilizationPercentage: 50
```

В этом YAML `scaleTargetRef` указывает объект нагрузки для масштабирования, а `targetCPUUtilizationPercentage` задает метрику триггера по CPU.

HPA с пользовательскими метриками

Для использования пользовательских метрик необходимо установить prometheus-operator и custom-metrics-api. После установки custom-metrics-api предоставляет множество ресурсов пользовательских метрик:

```
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "custom.metrics.k8s.io/v1beta1",
  "resources": [
    {
      "name": "namespaces/go_memstats_heap_sys_bytes",
      "singularName": "",
      "namespaced": false,
      "kind": "MetricValueList",
      "verbs": ["get"]
    },
    {
      "name": "jobs.batch/go_memstats_last_gc_time_seconds",
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": ["get"]
    },
    {
      "name": "pods/go_memstats_frees",
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": ["get"]
    }
  ]
}
```


Эти ресурсы являются подресурсами MetricValueList. Вы можете создавать правила через Prometheus для создания или поддержки подресурсов. Формат YAML HPA для пользовательских метрик отличается от традиционного HPA:

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: demo
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: demo
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Pods
      pods:
        metricName: metric-demo
        targetAverageValue: 10
```

В этом примере `scaleTargetRef` указывает нагрузку.

Определение условий триггера

- `metrics` — массив, поддерживающий несколько метрик
- `тип метрики` может быть: Object (описание ресурсов k8s), Pods (метрики для каждого Pod), Resources (встроенные метрики k8s: CPU, память), или External (обычно метрики вне кластера)
- Если пользовательская метрика не предоставляется Prometheus, необходимо создать новую метрику через ряд операций, например, создание правил в Prometheus

Основная структура метрики выглядит так:

```
{
      "describedObject": {  # Описываемый объект (Pod)
        "kind": "Pod",
        "namespace": "monitoring",
        "name": "nginx-788f78d959-fd6n9",
        "apiVersion": "/v1"
      },
      "metricName": "metric-demo",
      "timestamp": "2020-02-5T04:26:01Z",
      "value": "50"
}
```

Эти данные метрики собираются и обновляются Prometheus.

Совместимость HPA с пользовательскими метриками

YAML HPA с пользовательскими метриками совместим с оригинальными core metrics (CPU). Пример записи:

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: nginx
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        targetAverageUtilization: 80
    - type: Resource
      resource:
        name: memory
        targetAverageValue: 200Mi
```

- `targetAverageValue` — среднее значение, полученное для каждого пода
- `targetAverageUtilization` — использование, рассчитанное из прямого значения

Алгоритм расчёта:

```
replicas = ceil(sum(CurrentPodsCPUUtilization) / Target)
```

Обновления в autoscaling/v2beta2

autoscaling/v2beta2 поддерживает использование памяти:

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
  namespace: default
spec:
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - resource:
        name: cpu
        target:
          averageUtilization: 70
          type: Utilization
      type: Resource
    - resource:
        name: memory
        target:
          averageUtilization:
          type: Utilization
      type: Resource
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
```

Изменения: `targetAverageUtilization` и `targetAverageValue` заменены на `target` и преобразованы в комбинацию `xxxValue` и `type`:

- `xxxValue`: AverageValue (среднее значение), AverageUtilization (среднее использование), Value (прямое значение)
- `type`: Utilization (использование), AverageValue (среднее значение)

Примечания:

- Для метрик **CPU Utilization** и **Memory Utilization** автоскейлинг срабатывает только при отклонении фактического значения за пределы ±10% от целевого порога.
- Масштабирование вниз может повлиять на текущие бизнес-процессы; действуйте с осторожностью.

Правила расчёта

При изменении бизнес-метрик платформа автоматически рассчитывает целевое количество подов, соответствующее объему бизнеса, согласно следующим правилам и корректирует масштабирование. Если бизнес-метрики продолжают колебаться, значение будет ограничено установленным **Минимальным количеством подов** или **Максимальным количеством подов**.

- Целевое количество подов для одной политики: ceil[(сумма фактических значений метрик) / (порог метрики)] . Это означает, что сумма фактических значений метрик всех подов делится на порог метрики, и результат округляется вверх до ближайшего целого числа. Например: если сейчас 3 пода с использованием CPU 80%, 80% и 90%, а установлен порог CPU 60%, то по формуле количество подов будет автоматически скорректировано до: ceil[(80%+80%+90%) / 60%] = ceil 4.1 = 5 подов.
  Примечание:
  - Если рассчитанное целевое количество подов превышает установленное **Максимальное количество подов** (например, *4*), платформа масштабирует только до 4 подов. Если после изменения максимума метрики остаются высокими, возможно, потребуется использовать альтернативные методы масштабирования, например, увеличить квоту подов в namespace или добавить аппаратные ресурсы.
  - Если рассчитанное целевое количество подов (в примере *5*) меньше количества подов, рассчитанного с учётом **Шага масштабирования вверх** (например, *10*), платформа масштабирует только до 5 подов.

- Целевое количество подов для нескольких политик: берется максимальное значение из результатов расчётов каждой политики.

Запуск и остановка приложений

Содержание

Запуск приложения

1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Application** > **Applications**.
3. Нажмите на название приложения.
4. Нажмите **Start**.

Остановка приложения

1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Application** > **Applications**.
3. Нажмите на название приложения.
4. Нажмите **Stop**.

5. Ознакомьтесь с сообщением-подтверждением и, убедившись, что всё верно, нажмите **Stop**.

Настройка VerticalPodAutoscaler (VPA)

Содержание

Обзор

Alauda Container Platform Vertical Pod Autoscaler основан на открытом компоненте Vertical Pod Autoscaler. Он анализирует исторические данные об использовании ресурсов подами и предоставляет рекомендации по квотам для улучшения использования ресурсов.

Для как stateless, так и stateful приложений VerticalPodAutoscaler (VPA) автоматически рекомендует — и при необходимости применяет — более подходящие лимиты CPU и памяти на основе потребностей вашего приложения. Это помогает обеспечить подам достаточные ресурсы и одновременно улучшить общее использование ресурсов в кластере.

Примечания к выпуску

В этом разделе описаны новые функции, улучшения, устаревшие возможности и известные проблемы плагина **Alauda Container Platform Vertical Pod Autoscaler**.

Поддерживаемые версии

Версия	Версия Alauda Container Platform
v4.2.0	**v4.0**, **v4.1**, **v4.2**

Примечания к выпуску v4.2

v4.2.0

1. Исправление уязвимостей безопасности.
2. Теперь может выпускаться независимо.

Политика жизненного цикла

В следующей таблице приведён график жизненного цикла выпущенных версий плагина **Alauda Container Platform Vertical Pod Autoscaler**:

Версия	Дата выпуска	Дата окончания поддержки
v4.2.0	2025-12-16	2027-12-16

Понимание VerticalPodAutoscalers

VerticalPodAutoscaler (VPA) используется для рекомендации или автоматического обновления запросов и лимитов CPU и памяти для ваших подов на основе их исторических шаблонов использования.

После создания VPA платформа начинает отслеживать использование ресурсов целевых подов. Как только собирается достаточное количество данных, VPA вычисляет рекомендуемые значения ресурсов. В зависимости от настроенного режима обновления VPA может либо автоматически применять эти рекомендации, либо просто предоставлять их для ручного просмотра и применения.

Анализируя использование ресурсов с течением времени, VPA помогает обеспечить выделение подам необходимых ресурсов без избыточного резервирования, что ведёт к более эффективному использованию ресурсов всего кластера.

Как работает VPA?

VerticalPodAutoscaler (VPA) расширяет концепцию оптимизации ресурсов подов. VPA отслеживает использование ресурсов ваших подов и предоставляет рекомендации по запросам CPU и памяти на основе наблюдаемых шаблонов использования.

VPA работает, постоянно отслеживая использование ресурсов подов и обновляя свои рекомендации по мере поступления новых данных. VPA может работать в следующих режимах:

- **Off**: VPA только предоставляет рекомендации без их автоматического применения.
- **Manual Adjustment**: Вы можете вручную корректировать конфигурации ресурсов на основе рекомендаций VPA.

> **Важно**: Эластичное масштабирование (горизонтальное или вертикальное) работает лучше всего при достаточном количестве ресурсов в кластере. При нехватке ресурсов

> действия масштабирования могут привести к тому, что поды останутся в состоянии `Pending`. Убедитесь, что в вашем кластере достаточно ресурсов, установлены разумные квоты и настроены оповещения для мониторинга событий масштабирования.

Поддерживаемые функции

VPA предоставляет рекомендации по ресурсам на основе исторических шаблонов использования, позволяя оптимизировать конфигурации CPU и памяти ваших подов.

> **Важно**: При ручном применении рекомендаций VPA происходит пересоздание подов, что может вызвать временные перебои в работе приложения. Рассмотрите возможность применения рекомендаций в окна обслуживания для рабочих нагрузок в продакшене.

Предварительные требования

Перед использованием VPA убедитесь в следующем:

- В вашем кластере установлен кластерный плагин **Alauda Container Platform Vertical Pod Autoscaler**.
  - Скачайте последний пакет плагина, совместимый с вашей версией платформы.
  - Используйте CLI-инструмент `violet` для загрузки пакетов **Alauda Container Platform Vertical Pod Autoscalers** и **Alauda DevOps Pipelines** в целевой кластер. Подробные инструкции по использованию `violet` доступны в разделе CLI.
- Убедитесь, что компоненты мониторинга развернуты в текущем кластере и работают корректно. Вы можете проверить статус развертывания и состояние компонентов мониторинга, нажав в правом верхнем углу платформы ? > **Platform Health Status**..

Установка плагина Vertical Pod Autoscaler

1. Войдите в систему и перейдите на страницу **Administrators**.
2. Нажмите **Marketplace** > **Cluster Plugins**, чтобы открыть список **Cluster Plugins**.

3. Найдите кластерный плагин Alauda Container Platform Vertical Pod Autoscaler, нажмите **Install** и перейдите на страницу установки.

Обновление плагина Vertical Pod Autoscaler

1. Войдите в систему и перейдите на страницу **Administrators**.
2. Нажмите **Marketplace** > **Cluster Plugins**, чтобы открыть список **Cluster Plugins**.
3. Найдите кластерный плагин Alauda Container Platform Vertical Pod Autoscaler, нажмите **Upgrade** и перейдите на страницу установки.

Создание VerticalPodAutoscaler

Использование CLI

Вы можете создать VerticalPodAutoscaler с помощью командной строки, определив YAML-файл и используя команду `kubectl create`. В следующем примере показано вертикальное автоскейлирование подов для объекта Deployment:

1. Создайте YAML-файл с именем `vpa.yaml` со следующим содержимым:

```yaml
apiVersion: autoscaling.k8s.io/v1 1
kind: VerticalPodAutoscaler 2
metadata:
  name: my-deployment-vpa 3
  namespace: default
spec:
  targetRef:
    apiVersion: apps/v1 4
    kind: Deployment 5
    name: my-deployment 6
  updatePolicy:
    updateMode: 'Off' 7
  resourcePolicy: 8
    containerPolicies:
      - containerName: '*' 9
        mode: 'Auto' 10
```

1. Используйте API autoscaling.k8s.io/v1.
2. Имя VPA.
3. Укажите целевой объект рабочей нагрузки. VPA использует селектор рабочей нагрузки для поиска подов, которым нужно скорректировать ресурсы. Поддерживаемые типы рабочих нагрузок: DaemonSet, Deployment, ReplicaSet, StatefulSet, ReplicationController, Job и CronJob.
4. Укажите версию API объекта для масштабирования.
5. Укажите тип объекта.
6. Целевой ресурс, к которому применяется VPA.
7. Политика обновления, определяющая, как VPA применяет рекомендации. Параметр updateMode может принимать значения:

   - Auto: Автоматически устанавливает запросы ресурсов при создании подов и обновляет текущие поды до рекомендуемых значений. В настоящее время эквивалентно "Recreate". Этот режим может вызвать простой приложения. После поддержки обновлений ресурсов подов на месте режим "Auto" будет использовать этот механизм.
   - Recreate: Автоматически устанавливает запросы ресурсов при создании подов и эвакуирует текущие поды для обновления до рекомендуемых значений. Обновления на месте не используются.
   - Initial: Устанавливает запросы ресурсов только при создании подов, последующие изменения не применяются.
   - Off: Не изменяет запросы ресурсов подов автоматически, только предоставляет рекомендации в объекте VPA.

8. Политика ресурсов, позволяющая задавать отдельные стратегии для разных контейнеров. Например, установка режима контейнера в "Auto" означает, что для этого контейнера будут рассчитываться рекомендации, а "Off" — что рекомендации не будут рассчитываться.
9. Применить политику ко всем контейнерам в поде.
10. Установить режим Auto или Off. Auto означает, что для этого контейнера будут генерироваться рекомендации, Off — что рекомендации не будут генерироваться.

2. Примените YAML-файл для создания VPA:

```
kubectl create -f vpa.yaml
```

Пример вывода:

```
verticalpodautoscaler.autoscaling.k8s.io/my-deployment-vpa created
```

3. После создания VPA вы можете просмотреть рекомендации, выполнив команду:

```
kubectl describe vpa my-deployment-vpa
```

Пример вывода (частичный):

```
Status:
  Recommendation:
    Container Recommendations:
      Container Name:  my-container
      Lower Bound:
        Cpu:     100m
        Memory:  262144k
      Target:
        Cpu:     200m
        Memory:  524288k
      Upper Bound:
        Cpu:     300m
        Memory:  786432k
```

Использование веб-консоли

1. Войдите в систему и перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Workloads** > **Deployments**.
3. Кликните по ***Имя Deployment***.
4. Прокрутите вниз до раздела **Elastic Scaling** и справа нажмите **Update**.
5. Выберите **Vertical Scaling** и настройте правила масштабирования.

Параметр	Описание
Режим масштабирования	В настоящее время поддерживается режим **Manual Scaling**, который предоставляет рекомендуемые конфигурации ресурсов на основе анализа прошлых данных об использовании ресурсов. Вы можете вручную корректировать значения согласно рекомендациям. Корректировки приведут к пересозданию и перезапуску подов, поэтому выбирайте подходящее время, чтобы не повлиять на работающие приложения. Обычно после работы подов более 8 дней рекомендуемые значения становятся точными. Обратите внимание, что при недостатке ресурсов в кластере масштабирование может привести к состоянию Pending у подов. Убедитесь, что в кластере достаточно ресурсов или установлены разумные квоты, либо настройте оповещения для мониторинга условий масштабирования.
Целевой контейнер	По умолчанию выбран первый контейнер рабочей нагрузки. Вы можете включить рекомендации по лимитам ресурсов для одного или нескольких контейнеров по необходимости.

6. Нажмите **Update**.

Расширенная настройка VPA

Опции политики обновления

- `updateMode: "Off"` — VPA только предоставляет рекомендации без их автоматического применения. Вы можете применять рекомендации вручную по мере необходимости.
- `updateMode: "Auto"` — Автоматически устанавливает запросы ресурсов при создании подов и обновляет текущие поды до рекомендуемых значений. В настоящее время эквивалентно "Recreate".
- `updateMode: "Recreate"` — Автоматически устанавливает запросы ресурсов при создании подов и эвакуирует текущие поды для обновления до рекомендуемых

значений.

- `updateMode: "Initial"` — Устанавливает запросы ресурсов только при создании подов, последующие изменения не применяются.
- `minReplicas: <number>` — Минимальное количество реплик. Обеспечивает, что при эвакуации подов Updater не уменьшит количество доступных подов ниже этого значения. Должно быть больше 0.

Опции политики контейнера

- `containerName: "*"` — Применить политику ко всем контейнерам в поде.
- `mode: "Auto"` — Автоматически генерировать рекомендации для контейнера.
- `mode: "Off"` — Не генерировать рекомендации для контейнера.

Примечания:

- Рекомендации VPA основаны на исторических данных использования, поэтому может потребоваться несколько дней работы подов, прежде чем рекомендации станут точными.
- При применении рекомендаций VPA в режиме Auto происходит пересоздание подов, что может вызвать временные перебои в работе приложения.

Последующие действия

После настройки VPA рекомендуемые значения лимитов CPU и памяти для целевого контейнера можно просмотреть в разделе **Elastic Scaling**. В области **Containers** выберите вкладку целевого контейнера и нажмите значок справа от **Resource Limits**, чтобы обновить лимиты ресурсов согласно рекомендуемым значениям.

Настройка CronHPA

Для безсостоянных приложений с периодическими колебаниями бизнес-нагрузки CronHPA (Cron Horizontal Pod Autoscaler) поддерживает регулирование количества подов на основе заданных вами временных политик, что позволяет оптимизировать использование ресурсов в соответствии с предсказуемыми бизнес-паттернами.

Содержание

Понимание Cron Horizontal Pod Autoscalers

Вы можете создать cron horizontal pod autoscaler, чтобы указать количество подов, которые должны работать в определённое время согласно расписанию, что позволяет подготовиться к предсказуемым пиковым нагрузкам или сократить использование ресурсов в часы пониженной активности.

После создания cron horizontal pod autoscaler платформа начинает отслеживать расписание и автоматически регулирует количество подов в указанные моменты времени. Такое масштабирование по времени происходит независимо от метрик использования ресурсов, что делает его идеальным для приложений с известными паттернами использования.

CronHPA работает путём определения одного или нескольких правил расписания, каждое из которых указывает время (в формате crontab) и целевое количество реплик. Когда наступает запланированное время, CronHPA изменяет количество подов в соответствии с указанной целью, независимо от текущего использования ресурсов.

Как работает CronHPA?

Cron horizontal pod autoscaler (CronHPA) расширяет концепцию автоскейлинга подов, добавляя управление на основе времени. CronHPA позволяет определить конкретные моменты времени, когда количество подов должно изменяться, что помогает подготовиться к предсказуемым пиковым нагрузкам или снизить использование ресурсов в часы пониженной активности.

CronHPA постоянно сравнивает текущее время с заданными расписаниями. Когда наступает запланированное время, контроллер изменяет количество подов так, чтобы оно соответствовало целевому количеству реплик, указанному для этого расписания. Если несколько расписаний срабатывают одновременно, платформа использует правило с более высоким приоритетом (то есть то, которое определено раньше в конфигурации).

Предварительные требования

Убедитесь, что компоненты мониторинга развернуты в текущем кластере и работают корректно. Вы можете проверить статус развертывания и состояние компонентов мониторинга, кликнув в правом верхнем углу платформы ? > **Platform Health Status**..

Создание Cron Horizontal Pod Autoscaler

Использование CLI

Вы можете создать cron horizontal pod autoscaler через интерфейс командной строки, определив YAML-файл и используя команду `kubectl create`. В следующем примере показано плановое масштабирование для объекта Deployment:

1. Создайте YAML-файл с именем `cronhpa.yaml` со следующим содержимым:

```yaml
apiVersion: tkestack.io/v1 (1)
kind: CronHPA (2)
metadata:
  name: my-deployment-cronhpa (3)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (4)
    kind: Deployment (5)
    name: my-deployment (6)
  crons:
    - schedule: '0 0 * * *' (7)
      targetReplicas: 0 (8)
    - schedule: '0 8 * * 1-5' (9)
      targetReplicas: 3 (10)
    - schedule: '0 18 * * 1-5' (11)
      targetReplicas: 1 (12)
```

(1) Используйте API версии tkestack.io/v1.

(2) Имя ресурса CronHPA.

(3) Имя деплоймента для масштабирования.

(4) Укажите API версию объекта для масштабирования.

(5) Укажите тип объекта. Объект должен быть Deployment, ReplicaSet или StatefulSet.

(6) Целевой ресурс, к которому применяется CronHPA.

(7) Расписание в стандартном формате crontab (минута час день месяц день_недели).

(8) Целевое количество реплик для масштабирования при срабатывании расписания.

Этот пример настраивает деплоймент следующим образом:

- Масштабировать до 0 реплик в полночь каждый день
- Масштабировать до 3 реплик в 8:00 утра по будням (понедельник-пятница)
- Масштабировать до 1 реплики в 18:00 по будням

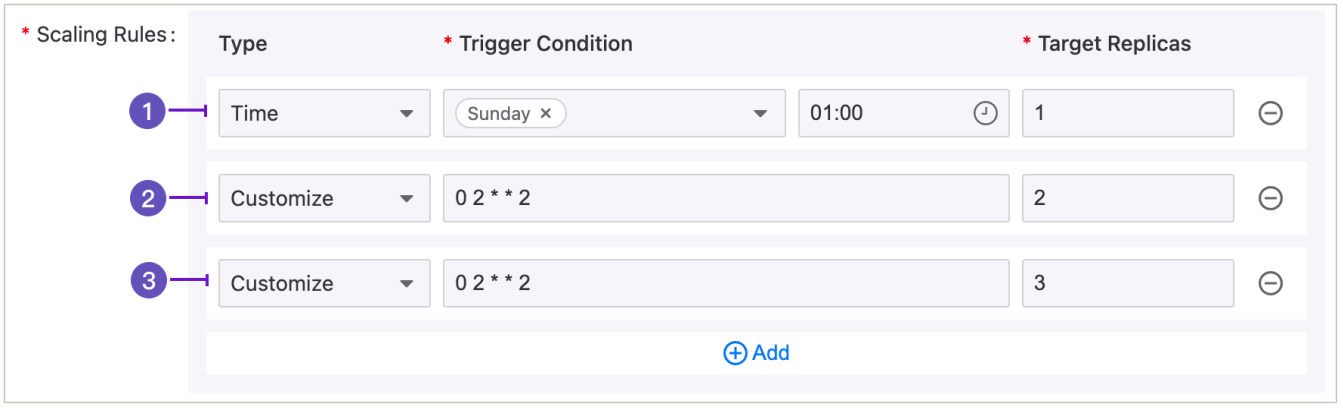
2. Примените YAML-файл для создания CronHPA:

```
kubectl create -f cronhpa.yaml
```

Использование веб-консоли

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Workloads** > **Deployments**.
3. Кликните на ***Имя Deployment***.
4. Пролистайте вниз до раздела **Elastic Scaling** и нажмите **Update** справа.
5. Выберите **Scheduled Scaling** и настройте правила масштабирования. Если выбран тип **Custom**, необходимо указать выражение Crontab для условия срабатывания в формате `минута час день месяц неделя`. Для подробного описания обратитесь к разделу Writing Crontab Expressions.
6. Нажмите **Update**.

Объяснение правил расписания



1. Указывает, что начиная с 01:00 каждого понедельника будет оставаться только 1 под.
2. Указывает, что начиная с 02:00 каждого вторника будет оставаться только 2 пода.

3. Указывает, что начиная с 02:00 каждого вторника будет оставаться только 3 пода.

Важные замечания:

- Если несколько правил имеют одинаковое время срабатывания (примеры 2 и 3), платформа выполнит автоматическое масштабирование только по правилу с более высоким приоритетом (пример 2).
- CronHPA работает независимо от HPA. Если оба настроены для одной и той же нагрузки, они могут конфликтовать. Тщательно продумайте стратегию масштабирования.
- Расписание использует формат crontab (`минута час день месяц неделя`) и следует тем же правилам, что и Kubernetes CronJobs.
- Время основывается на часовом поясе кластера.
- Для нагрузок с критическими требованиями к доступности убедитесь, что запланированное масштабирование не приведёт к неожиданному снижению мощности в периоды высокой нагрузки.

Обновление приложений

Пользовательские приложения значительно упрощают единое управление рабочими нагрузками, сетями, хранилищами и конфигурациями, но не все ресурсы принадлежат приложению.

- Ресурсы, добавленные в процессе создания приложения или при обновлении приложения, по умолчанию ассоциируются с приложением и не требуют дополнительного импорта.
- Ресурсы, созданные вне приложения, не принадлежат приложению и не отображаются в деталях приложения. Однако, если определения ресурсов соответствуют бизнес-требованиям, бизнес может функционировать нормально. В этом случае рекомендуется импортировать ресурсы в приложение для единого управления.
- **Управление образами**
  - Развертывание новых контейнерных образов с контролем тегов/патч-версий
  - Настройка imagePullPolicy (Always/IfNotPresent/Never)
- **Конфигурация времени выполнения**
  - Изменение переменных окружения через ConfigMaps/Secrets
  - Обновление запросов/лимитов ресурсов (CPU/Память)
- **Оркестрация ресурсов**
  - Импорт существующих Kubernetes-ресурсов (Deployments/Services/Ingresses)
  - Синхронизация конфигураций между неймспейсами с помощью `kubectl apply -f`

Импортированные в приложение ресурсы получают следующие преимущества:

Функция	Описание
Снимок версии	При создании снимка версии для приложения также создаётся снимок ресурсов внутри приложения. • Если приложение откатывается, ресурсы также откатываются к состоянию из снимка. • Если распространяется конкретная версия приложения, платформа автоматически создаст ресурсы, зафиксированные в снимке, при повторном развертывании приложения.
Удаление вместе с приложением	Если приложение больше не нужно, удаление приложения автоматически удалит все ресурсы, связанные с приложением, включая вычислительные компоненты, внутренние маршруты и входящие правила.
Проще найти	В деталях приложения можно быстро просмотреть ресурсы, связанные с приложением. Например: Внешний трафик может получить доступ к *Deployment D* через *Service S*, который принадлежит *Application A*, но соответствующий адрес доступа можно быстро найти в деталях приложения только если *Service S* также принадлежит *Application A*.

Содержание

Импорт ресурсов

> Пакетный импорт связанных ресурсов в неймспейсе, где находится приложение; ресурс может принадлежать только одному приложению.

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***Название приложения***.
4. Нажмите **Actions** > **Manage Resources**.
5. Внизу в разделе **Resource Type** выберите тип ресурсов для импорта.
   Примечание: Распространённые типы ресурсов включают Deployment, DaemonSet, StatefulSet, Job, CronJob, Service, Ingress, PVC, ConfigMap, Secret и HorizontalPodAutoscaler, которые отображаются вверху; остальные ресурсы расположены в алфавитном порядке, и вы можете быстро найти нужный тип ресурса, введя ключевые слова.
6. В разделе **Resources** выберите ресурсы для импорта.
   Внимание: Для ресурсов типа **Job** поддерживается импорт только задач, созданных через YAML.
7. Нажмите **Import Resources**.

Удаление/пакетное удаление ресурсов

> Удаление/пакетное удаление ресурсов из приложения лишь разрывает связь приложения с ресурсами и не удаляет сами ресурсы.

Если между ресурсами в приложении существуют взаимосвязи, удаление любого ресурса из приложения не изменит связи между ресурсами. Например, даже если *Service S* удалён из *Application A*, внешний трафик всё равно может получить доступ к *Deployment D* через *Service S*.

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***Название приложения***.
4. Нажмите **Actions** > **Manage Resources**.
5. Нажмите **Remove** справа от ресурса, чтобы удалить его; либо выберите несколько ресурсов одновременно и нажмите **Remove** вверху таблицы для пакетного удаления

ресурсов.

Экспорт приложений

Для стандартизации процесса экспорта приложений между средами разработки, тестирования и продакшена, а также для облегчения быстрой миграции бизнеса в новые среды, вы можете экспортировать нативные приложения в виде шаблонов приложений (Charts) или экспортировать упрощённые YAML-файлы, которые можно использовать напрямую для развертывания. Это позволяет запускать нативное приложение в разных средах или пространствах имён. Также вы можете экспортировать YAML-файлы в репозиторий кода для быстрого развертывания приложений в кластерах с помощью функционала GitOps.

Содержание

Экспорт Helm Charts

Процедура

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***название приложения*** типа `Custom Application`.
4. Нажмите **Actions** > **Export**; также можно экспортировать конкретную версию на странице деталей приложения.
5. Выберите нужный метод экспорта и следуйте инструкциям для настройки соответствующей информации.

   - Экспорт Helm Charts в репозиторий шаблонов с правами управления
     Примечание: Репозиторий шаблонов добавляется администратором платформы. Обратитесь к администратору платформы, чтобы получить действующий репозиторий шаблонов типа **Chart** или **OCI Chart** с правами **Management**.

Параметр	Описание
Целевое расположение	Выберите **Template Repository** для прямой синхронизации шаблона в репозиторий шаблонов типа **Chart** или **OCI Chart** с правами **Management**. Владелец проекта, назначенный для этого **Template Repository**, сможет напрямую использовать шаблон.
Каталог шаблона	Если выбран репозиторий шаблонов типа OCI Chart, необходимо выбрать или вручную ввести каталог для хранения Helm Chart. **Примечание**: При ручном вводе нового каталога платформа создаст этот каталог в репозитории шаблонов, но существует риск неудачи создания.
Версия	Номер версии шаблона приложения. Формат должен быть `v<Major>.<Minor>.<Patch>`. Значение по

Параметр	Описание
	умолчанию — текущая версия приложения или текущая версия снимка.
Иконка	Поддерживаются форматы изображений JPG, PNG и GIF, размер файла не более 500KB. Рекомендуемые размеры — 80*60 пикселей.
Описание	Описание будет отображаться в списке шаблонов приложений в каталоге приложений.
README	Файл описания. Поддерживается редактирование в формате Markdown, отображается на странице деталей шаблона приложения.
NOTES	Файл помощи шаблона. Поддерживается редактирование в виде обычного текста; после завершения шаблона развертывания он отображается на странице деталей шаблона приложения.

- Экспорт Helm Charts локально для последующей ручной загрузки в репозиторий шаблонов: выберите **Local** в качестве целевого расположения и формат файла **Helm Chart** для генерации пакета Helm Chart, который будет загружен локально для офлайн-передачи.

6. Нажмите **Export**.

Последующие действия

- Если вы экспортировали Helm Chart локально, вам потребуется добавить шаблон в репозиторий шаблонов с правами управления.
- Независимо от выбранного метода экспорта, вы можете обратиться к Созданию нативных приложений — метод шаблона для создания нативного приложения типа `Template Application` в **не текущем** пространстве имён.

Экспорт YAML локально

Шаги

Метод 1

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***название приложения***.
4. Нажмите **Actions** > **Export**; также можно экспортировать конкретную версию на странице деталей приложения.
5. Выберите **Local** в качестве целевого расположения и формат файла **YAML** — в этот момент можно экспортировать упрощённый YAML-файл, который можно развернуть напрямую в других средах.
6. Нажмите **Export**.

Метод 2

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***название приложения***.
4. Перейдите на вкладку **YAML**, настройте параметры по необходимости и просмотрите YAML-файл.

Тип	Описание
Полный YAML	По умолчанию **Preview Simplified YAML** не выбран, отображается YAML-файл с **скрытыми полями managedFields**. Вы можете просмотреть и экспортировать его напрямую; также можно снять галочку с **Hide managedFields fields** для экспорта полного YAML-файла. **Примечание**: Полный YAML в основном используется для операций

Тип	Описание
	и устранения неполадок и не подходит для быстрого создания нативных приложений на платформе.
Упрощённый YAML	При выборе **Preview Simplified YAML** можно просмотреть и экспортировать упрощённый YAML-файл, который можно развернуть напрямую в других средах.

5. Нажмите **Export**.

Последующие действия

После экспорта упрощённого YAML вы можете обратиться к Созданию нативных приложений — метод YAML для создания нативного приложения типа `Custom Application` в **не текущем** пространстве имён.

Экспорт YAML в репозиторий кода (Alpha)

Меры предосторожности

- Только администраторы платформы и администраторы проектов могут напрямую экспортировать YAML-файлы нативных приложений в репозиторий кода.
- `Template Applications` не поддерживают экспорт файлов конфигурации приложений в формате Kustomize или прямой экспорт YAML-файлов в репозиторий кода; вы можете сначала **отвязать от шаблона** и преобразовать в `Custom Application`.

Шаги

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***название приложения*** типа `Custom`.

4. Нажмите **Actions** > **Export**; также можно экспортировать конкретную версию на странице деталей приложения.
5. Выберите нужный метод экспорта и следуйте инструкциям для настройки соответствующей информации.

   - Экспорт YAML в репозиторий кода:

Параметр	Описание
Целевое расположение	Выберите **Code Repository** для прямой синхронизации YAML-файла в указанный Git-репозиторий. Владелец проекта, назначенный для этого **Code Repository**, сможет напрямую использовать YAML-файл.
Имя проекта интеграции	Имя проекта интеграционного инструмента, назначенное или связанное с вашим проектом администратором платформы.
Адрес репозитория	Адрес репозитория, назначенный для вашего использования в рамках проекта интеграционного инструмента.

Метод экспорта	• **Существующая ветка**: экспорт YAML приложения в выбранную ветку.
	• **Новая ветка**: создание новой ветки на основе выбранной **Branch/Tag/Commit ID** и экспорт YAML приложения в новую ветку.
	  • Если отмечен пункт **Submit PR (Pull Request)**, платформа создаст новую ветку и отправит Pull Request.
	  • Если отмечен пункт **Automatically delete source branch after merging PR**, исходная ветка будет автоматически удалена после слияния PR в Git-репозитории.

Параметр	Описание
Путь к файлу	Конкретное расположение файла в репозитории кода; можно также ввести путь к файлу, и платформа создаст новый путь в репозитории на основе введённого.
Сообщение коммита	Заполните информацию коммита для идентификации содержимого этого отправления.
Предварительный просмотр	Просмотр YAML-файла для отправки и сравнение с существующим YAML в репозитории кода с цветовой подсветкой различий.

- Экспорт файлов типа Kustomize локально для последующей ручной загрузки в репозиторий кода: выберите **Local** в качестве целевого расположения и формат файла **Kustomize** для локального экспорта конфигурационного файла приложения типа Kustomize. Этот файл поддерживает дифференцированные конфигурации и подходит для развертывания приложений в разных кластерах.

6. Нажмите **Export**.

Последующие действия

После экспорта YAML в Git-репозиторий вы можете обратиться к Creating GitOps Applications↗ для создания GitOps-приложения типа `Custom Application` для кросс-кластерного развертывания.

Обновление и удаление Chart-приложений

В связи с пересечением функционала между текущими шаблонными приложениями и нативными приложениями, а также с расширенными операционными возможностями, доступными в нативных приложениях, независимое управление шаблонными приложениями в будущих версиях больше не будет поддерживаться. Пожалуйста, как можно скорее обновите успешно развернутые шаблонные приложения до нативных приложений.

Содержание

Важные замечания

Эта функция **будет прекращена**. Пожалуйста, как можно скорее обновите успешно развернутые шаблонные приложения до нативных приложений.

Предварительные требования

Пожалуйста, свяжитесь с администратором платформы для включения функций, связанных с шаблонными приложениями.

Описание анализа статуса

Нажмите на ***Template Application Name***, чтобы отобразить подробный анализ статуса развертывания Chart в детальной информации.

Тип	Причина

Initialized	Указывает состояние загрузки шаблона Chart.
	• Если статус True, это означает, что загрузка шаблона Chart прошла успешно.
	• Если статус False, это означает, что загрузка шаблона Chart не удалась, а причину неудачи можно посмотреть в столбце сообщения.
	  • ChartLoadFailed: загрузка шаблона Chart не удалась.
	  • InitializeFailed: во время инициализации перед загрузкой Chart произошла ошибка.

Validated	Указывает состояние проверки прав пользователя и зависимостей для шаблона Chart.
	• Если статус True, это означает, что все проверки прошли успешно.
	• Если статус False, это означает, что некоторые проверки не прошли, а причину неудачи можно посмотреть в столбце сообщения.
	  • DependenciesCheckFailed: проверка зависимостей Chart не удалась.
	  • PermissionCheckFailed: у текущего пользователя нет прав на выполнение некоторых операций с ресурсами.
	  • ConsistentNamespaceCheckFailed: при развертывании шаблонного приложения как нативного приложение Chart содержит ресурсы, требующие развертывания в разных пространствах имён.

Тип	Причина
Synced	Указывает состояние развертывания шаблона Chart. • Если статус True, это означает, что развертывание шаблона Chart прошло успешно. • Если статус False, это означает, что развертывание шаблона Chart не удалось, причина отображается как ChartSyncFailed, а конкретную причину неудачи можно посмотреть в столбце сообщения.

Управление версиями приложений

После обновления приложения через интерфейс платформы автоматически создаётся запись о версии в истории. Для обновлений приложения, инициированных не через интерфейс, например, при обновлении приложения через API-вызовы, вы можете вручную создать снимок версии для фиксации изменений.

Примечание: Когда количество записей снимков версий превышает 6, платформа сохраняет только последние 6 записей и автоматически удаляет остальные, отдавая приоритет удалению самых старых снимков версий.

Содержание

Создание снимка версии

Порядок действий

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.

3. Нажмите на ***Application Name***.
4. Во вкладке **Version Snapshot** нажмите **Create Version Snapshot**.
5. Заполните информацию и нажмите **Confirm**.
   Примечание: Вы также можете Distribute the Application, что позволяет распространять снимок версии приложения в виде Chart, облегчая быструю развертку одного и того же приложения на нескольких кластерах и пространствах имён платформы.

Откат к исторической версии

Выполните откат текущей конфигурации приложения к исторической версии.

Порядок действий

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Нажмите на ***Application Name***.
4. Во вкладке **Historical Versions** нажмите на ***Version Number***.
5. Нажмите ⋮ > **Roll Back to This Version**.
6. Нажмите **Roll Back**.

Удаление приложений

При удалении приложения одновременно удаляется само приложение и все Kubernetes-ресурсы, которые непосредственно входят в его состав. Кроме того, это действие разрывает любые связи приложения с другими Kubernetes-ресурсами, которые не были напрямую частью его определения.

Обработка ошибок нехватки ресурсов

Содержание

Overview

В этом руководстве описывается, как предотвратить исчерпание памяти (OOM) или дискового пространства на узлах Alauda Container Platform. Стабильная работа узла критически важна, особенно для несжимаемых ресурсов, таких как память и диск. Истощение ресурсов может привести к нестабильности узла.

Администраторы могут настроить политики эвакуации, чтобы отслеживать состояние узлов и освобождать ресурсы до того, как стабильность будет нарушена.

В этом документе рассматривается, как Alauda Container Platform обрабатывает ситуации нехватки ресурсов, включая освобождение ресурсов, эвакуацию подов, планирование подов и механизм Out of Memory Killer. Также приведены примеры конфигураций и рекомендации по лучшим практикам.

> **NOTE**
>
> Если на узле включена swap-память, давление по памяти обнаружить невозможно. Отключите swap, чтобы включить эвакуацию на основе памяти.

Настройка политик эвакуации

Политики эвакуации позволяют узлам завершать поды при нехватке ресурсов, освобождая необходимые ресурсы. Политики состоят из сигналов эвакуации и пороговых значений, которые задаются в конфигурации узла или через командную строку. Эвакуация может быть:

- **Жесткой (Hard)**: немедленное действие при превышении порога.
- **Мягкой (Soft)**: действие после периода ожидания.

Правильно настроенные политики эвакуации помогают узлам проактивно предотвращать исчерпание ресурсов.

> **NOTE**
>
> При эвакуации пода все контейнеры в нем завершаются, а состояние PodPhase переходит в Failed.

Для контроля давления на диск узлы мониторят как `nodefs` (корневая файловая система), так и `imagefs` (хранилище образов контейнеров).

- **nodefs/rootfs**: используется для локальных дисковых томов, логов и другого хранилища (например, `/var/lib/kubelet`).
- **imagefs**: используется контейнерным рантаймом для образов и изменяемых слоев.

> **NOTE**
>
> Без изоляции локального хранилища (ephemeral storage) или квот XFS (volumeConfig) ограничить использование диска подом невозможно.

Создание политик эвакуации в конфигурации узла

Для задания порогов эвакуации отредактируйте карту конфигурации узла в разделе `eviction-hard` или `eviction-soft`.

Пример жесткой эвакуации:

```
kubeletArguments:
  eviction-hard: (1)
    - memory.available<100Mi (2)
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
    - imagefs.inodesFree<10%
```

(1) Тип эвакуации: используйте `eviction-hard` для жестких порогов.

(2) Каждый порог эвакуации задается в формате `<eviction_signal><operator><quantity>`, например, `memory.available<500Mi` или `nodefs.available<10%`.

> **NOTE**

Для `inodesFree` используйте процентные значения. Другие параметры принимают проценты или числовые значения.

Пример мягкой эвакуации:

```
kubeletArguments:
  eviction-soft: 1
    - memory.available<100Mi 2
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
    - imagefs.inodesFree<10%
  eviction-soft-grace-period: 3
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

1 Тип эвакуации: используйте `eviction-soft` для мягких порогов.

2 Каждый порог эвакуации задается в формате `<eviction_signal><operator><quantity>`, например, `memory.available<500Mi` или `nodefs.available<10%`.

3 Период ожидания перед эвакуацией при мягком пороге. Для оптимальной работы оставьте значения по умолчанию.

Перезапустите службу kubelet, чтобы изменения вступили в силу:

```
$ systemctl restart kubelet
```

Сигналы эвакуации

Узлы могут инициировать эвакуацию на основе следующих сигналов:

Состояние узла	Сигнал эвакуации	Описание

MemoryPressure	memory.available	Доступная память ниже порога
DiskPressure	nodefs.available	Свободное пространство на корневой файловой системе узла ниже порога
	nodefs.inodesFree	Свободные индексы inode ниже порога
	imagefs.available	Свободное пространство на файловой системе образов ниже порога
	imagefs.inodesFree	Свободные индексы inode в imagefs ниже порога

- `inodesFree` должен задаваться в процентах.
- Расчеты памяти исключают освобождаемую неактивную файловую память.
- Не используйте `free -m` внутри контейнеров.

Узлы проверяют эти файловые системы каждые 10 секунд. Специализированные файловые системы для томов и логов не мониторятся.

> **NOTE**
>
> Перед эвакуацией подов из-за давления на диск узлы выполняют сборку мусора контейнеров и образов.

Пороговые значения эвакуации

Пороговые значения эвакуации запускают освобождение ресурсов. При достижении порога узел сообщает о состоянии давления, предотвращая планирование новых подов до освобождения ресурсов.

- **Жесткие пороги**: немедленное действие.
- **Мягкие пороги**: действие после периода ожидания.

Порог задается в формате:

```
<eviction_signal><operator><quantity>
```

Примеры:

- `memory.available<1Gi`
- `memory.available<10%`

Узлы оценивают пороги каждые 10 секунд.

Жесткие пороги эвакуации

Без периода ожидания; действие выполняется немедленно.

Пример:

```
kubeletArguments:
  eviction-hard:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<5%
    - imagefs.available<100Mi
    - imagefs.inodesFree<10%
```

Значения по умолчанию для жестких порогов эвакуации

```
kubeletArguments:
  eviction-hard:
    - memory.available<100Mi
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
```

Мягкие пороги эвакуации

Требуют периода ожидания. Опционально можно задать максимальный период завершения пода (`eviction-max-pod-grace-period`).

Пример:

```
kubeletArguments:
  eviction-soft:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<5%
    - imagefs.available<100Mi
    - imagefs.inodesFree<10%
  eviction-soft-grace-period:
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

Настройка доступных ресурсов для планирования

Контролируйте, сколько ресурсов узла доступно для планирования, задавая `system-reserved` для системных демонов. Эвакуация происходит только если поды превышают запрошенные ресурсы.

- **Capacity**: общий ресурс узла.
- **Allocatable**: ресурс, доступный для планирования.

Пример:

```
kubeletArguments:
  eviction-hard:
    - "memory.available<500Mi"
  system-reserved:
    - "memory=1.5Gi"
```

Определите подходящие значения с помощью API сводки узла.

Перезапустите kubelet для применения изменений:

```
$ systemctl restart kubelet
```

Предотвращение колебаний состояния узла

Чтобы избежать колебаний выше/ниже мягких порогов эвакуации, задайте `eviction-pressure-transition-period`:

Пример:

```
kubeletArguments:
  eviction-pressure-transition-period:
    - 5m
```

По умолчанию 5 минут. Перезапустите службы для применения.

Освобождение ресурсов на уровне узла

При достижении критериев эвакуации узлы освобождают ресурсы до эвакуации пользовательских подов.

- **С imagefs**:
  - Если достигнут порог `nodefs`: удаляются мертвые поды/контейнеры.
  - Если достигнут порог `imagefs`: удаляются неиспользуемые образы.
- **Без imagefs**:
  - Если достигнут порог `nodefs`: удаляются мертвые поды/контейнеры, затем неиспользуемые образы.

Эвакуация подов

Если порог и период ожидания достигнуты, поды эвакуируются до тех пор, пока сигнал не станет ниже порога.

Поды ранжируются для эвакуации по качеству обслуживания (QoS) и потреблению ресурсов.

Уровень QoS	Описание
Guaranteed	Сначала эвакуируются поды с наибольшим потреблением ресурсов.
Burstable	Сначала эвакуируются поды с наибольшим потреблением относительно запроса.
BestEffort	Сначала эвакуируются поды с наибольшим потреблением ресурсов.

Поды Guaranteed эвакуируются только если системные демоны превышают зарезервированные ресурсы или остались только Guaranteed поды.

Диск — ресурс с лучшими усилиями (best-effort); поды эвакуируются по одному для освобождения места, ранжируясь по QoS и использованию диска.

Качество обслуживания и Out of Memory Killer

Если происходит системное событие OOM до освобождения памяти, срабатывает OOM killer.

OOM-оценки задаются на основе QoS:

Уровень QoS	Значение oom_score_adj
Guaranteed	-998
Burstable	min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)
BestEffort	1000

OOM killer завершает контейнер с наивысшим баллом. Сначала завершаются контейнеры с низшим QoS и наибольшим потреблением памяти. Контейнеры могут быть

перезапущены согласно политике узла.

Планировщик и условия нехватки ресурсов

Планировщик учитывает состояние узла при размещении подов.

Состояние узла	Поведение планировщика
MemoryPressure	Поды BestEffort не планируются.
DiskPressure	Новые поды не планируются.

Пример сценария

Оператор хочет:

- Узел с 10Gi памяти.
- Зарезервировать 10% для системных демонов.
- Эвакуировать поды при 95% использовании.

Расчет:

- `capacity = 10Gi`
- `system-reserved = 1Gi`
- `allocatable = 9Gi`

Чтобы вызвать эвакуацию при доступной памяти ниже 10% в течение 30 секунд или сразу при 5%:

- `system-reserved = 2Gi`
- `allocatable = 8Gi`

Конфигурация:

```
kubeletArguments:
  system-reserved:
    - "memory=2Gi"
  eviction-hard:
    - "memory.available<.5Gi"
  eviction-soft:
    - "memory.available<1Gi"
  eviction-soft-grace-period:
    - "memory.available=30s"
```

Это предотвращает немедленное давление по памяти и эвакуацию после планирования.

Рекомендуемые практики

Daemon Sets и обработка нехватки ресурсов

Поды, созданные daemon set, сразу пересоздаются при эвакуации. Daemon set должны избегать подов с QoS BestEffort и использовать Guaranteed QoS, чтобы снизить риск эвакуации.

Проверки состояния

Содержание

Понимание проверок состояния

Обратитесь к официальной документации Kubernetes:

- Liveness, Readiness, and Startup Probes ↗
- Configure Liveness, Readiness and Startup Probes ↗

> В Kubernetes проверки состояния, также известные как пробы, являются критически важным механизмом для обеспечения высокой доступности и устойчивости ваших приложений. Kubernetes использует эти пробы для определения состояния здоровья и готовности ваших Pod'ов, что позволяет системе предпринимать соответствующие действия, такие как перезапуск контейнеров или маршрутизация трафика. Без правильной настройки проверок состояния Kubernetes не сможет надежно управлять жизненным циклом вашего приложения, что может привести к ухудшению качества сервиса или сбоям.

Kubernetes предлагает три типа проб:

- `livenessProbe`: Определяет, запущен ли контейнер. Если liveness probe не проходит, Kubernetes завершит Pod и перезапустит его согласно политике перезапуска.
- `readinessProbe`: Определяет, готов ли контейнер обслуживать трафик. Если readiness probe не проходит, Endpoint Controller удаляет Pod из списка Endpoint'ов сервиса до тех пор, пока проба не пройдет успешно.
- `startupProbe`: Специально проверяет, успешно ли запустилось приложение. Liveness и readiness пробы не выполняются, пока startup probe не пройдет успешно. Это особенно полезно для приложений с длительным временем запуска.

Правильная настройка этих проб необходима для создания надежных и самовосстанавливающихся приложений в Kubernetes.

Типы проб

Kubernetes поддерживает три механизма реализации проб:

HTTP `GET` действие

> Выполняет HTTP-запрос `GET` к IP-адресу Pod'а на указанном порту и пути. Проба считается успешной, если код ответа находится в диапазоне от 200 до 399.

- **Сценарии использования**: Веб-серверы, REST API или любое приложение, предоставляющее HTTP-эндпоинт.
- **Пример**:

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 15
  periodSeconds: 20
```

`exec` действие

> Выполняет указанную команду внутри контейнера. Проба считается успешной, если команда завершается с кодом 0.

- **Сценарии использования**: Приложения без HTTP-эндпоинтов, проверка внутреннего состояния приложения или выполнение сложных проверок, требующих специальных инструментов.
- **Пример**:

```yaml
readinessProbe:
  exec:
    command:
      - cat
      - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

TCP `Socket` действие

> Пытается открыть TCP-сокет на IP-адресе контейнера и указанном порту. Проба считается успешной, если TCP-соединение устанавливается.

- **Сценарии использования**: Базы данных, очереди сообщений или любое приложение, которое общается по TCP-порту, но может не иметь HTTP-эндпоинта.
- **Пример**:

```
startupProbe:
  tcpSocket:
    port: 3306
  initialDelaySeconds: 5
  periodSeconds: 10
  failureThreshold: 30
```

Лучшие практики

- **Liveness** vs. **Readiness**:
  - **Liveness**: Если ваше приложение не отвечает, лучше его перезапустить. При сбое Kubernetes перезапустит контейнер.
  - **Readiness**: Если приложение временно не может обслуживать трафик (например, подключается к базе данных), но может восстановиться без перезапуска, используйте Readiness Probe. Это предотвратит маршрутизацию трафика на нездоровый экземпляр.
- **Startup Probes для медленных приложений**: Используйте Startup Probes для приложений, которым требуется значительное время на инициализацию. Это предотвратит преждевременные перезапуски из-за сбоев Liveness Probe или проблемы с маршрутизацией трафика из-за сбоев Readiness Probe во время запуска.
- **Легковесные пробы**: Убедитесь, что эндпоинты проб легковесны и выполняются быстро. Они не должны включать тяжелые вычисления или внешние зависимости (например, вызовы к базе данных), которые могут сделать пробу ненадежной.
- **Содержательные проверки**: Проверки должны реально отражать состояние здоровья и готовности приложения, а не просто факт работы процесса. Например, для веб-сервера проверяйте, может ли он обслуживать базовую страницу, а не только открыт ли порт.
- **Настройка initialDelaySeconds**: Устанавливайте initialDelaySeconds так, чтобы дать приложению достаточно времени для запуска перед первой проверкой.

- **Настройка periodSeconds и failureThreshold**: Балансируйте между быстрым обнаружением сбоев и избеганием ложных срабатываний. Слишком частые пробы или слишком низкий failureThreshold могут привести к ненужным перезапускам или состояниям «не готов».
- **Логи для отладки**: Убедитесь, что ваше приложение логирует понятные сообщения, связанные с вызовами эндпоинтов проверок и внутренним состоянием, чтобы облегчить отладку сбоев проб.
- **Комбинирование проб**: Часто все три пробы (Liveness, Readiness, Startup) используются вместе для эффективного управления жизненным циклом приложения.

Пример YAML-файла

```yaml
spec:
  template:
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2 # Container image
          ports:
            - containerPort: 80 # Container exposed port
          startupProbe:
            httpGet:
              path: /startup-check
              port: 8080
            initialDelaySeconds: 0 # Обычно 0 для startup probes или очень маленькое значение
            periodSeconds: 5
            failureThreshold: 60 # Позволяет 60 * 5 = 300 секунд (5 минут) на запуск
          livenessProbe:
            httpGet:
              path: /healthz
              port: 8080
            initialDelaySeconds: 5 # Задержка 5 секунд после запуска Pod перед проверкой
            periodSeconds: 10 # Проверка каждые 10 секунд
            timeoutSeconds: 5 # Таймаут через 5 секунд
            failureThreshold: 3 # Считается нездоровым после 3 последовательных сбоев
          readinessProbe:
            httpGet:
              path: /ready
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 10
            timeoutSeconds: 5
            failureThreshold: 3
```

Параметры настройки проверок состояния через веб-консоль

Общие параметры

Параметры	Описание
Initial Delay	`initialDelaySeconds`: Период ожидания (в секундах) перед началом проверок. По умолчанию: `300`.
Period	`periodSeconds`: Интервал между проверками (1-120 с). По умолчанию: `60`.
Timeout	`timeoutSeconds`: Время ожидания ответа пробы (1-300 с). По умолчанию: `30`.
Success Threshold	`successThreshold`: Минимальное количество последовательных успешных проверок для отметки как здорового. По умолчанию: `0`.
Failure Threshold	`failureThreshold`: Максимальное количество последовательных неудач для срабатывания действия: - `0`: Отключает действия при ошибках - По умолчанию: `5` неудач → перезапуск контейнера.

Параметры, специфичные для протокола

Параметр	Поддерживаемые протоколы	Описание
Protocol	HTTP/HTTPS	Протокол проверки состояния
Port	HTTP/HTTPS/TCP	Целевой порт контейнера для проверки.
Path	HTTP/HTTPS	Путь эндпоинта (например, `/healthz`).
HTTP Headers	HTTP/HTTPS	Пользовательские заголовки (добавление пар ключ-значение).

Параметр	Поддерживаемые протоколы	Описание
Command	EXEC	Команда для проверки, выполняемая внутри контейнера (например, `sh -c "curl -I localhost:8080 | grep OK"`). **Примечание**: Экранируйте специальные символы и тестируйте команду.

Устранение неполадок при сбоях проб

Если статус Pod указывает на проблемы, связанные с пробами, выполните следующие действия для диагностики:

Проверьте события Pod'а

```
kubectl describe pod <pod-name>
```

Ищите события, связанные с LivenessProbe failed, ReadinessProbe failed или StartupProbe failed. Эти события часто содержат конкретные сообщения об ошибках (например, отказ соединения, HTTP 500, код выхода команды).

Просмотрите логи контейнера

```
kubectl logs <pod-name> -c <container-name>
```

Изучите логи приложения, чтобы увидеть ошибки или предупреждения в момент сбоя пробы. Возможно, приложение логирует причины, по которым эндпоинт проверки не отвечает корректно.

Проверьте эндпоинт пробы вручную

- **HTTP**: Если возможно, выполните `kubectl exec -it <pod-name> -- curl <probe-path>:<probe-port>` или `wget` внутри контейнера, чтобы увидеть реальный ответ.
- **Exec**: Запустите команду пробы вручную: `kubectl exec -it <pod-name> -- <command-from-probe>` и проверьте код выхода и вывод.
- **TCP**: Используйте `nc` (netcat) или `telnet` из другого Pod в той же сети или с хоста (если разрешено), чтобы проверить TCP-соединение: `kubectl exec -it <another-pod> -- nc -vz <pod-ip> <probe-port>`.

Проверьте конфигурацию пробы

- Тщательно проверьте параметры пробы (путь, порт, команду, задержки, пороги) в YAML Deployment/Pod. Частая ошибка — неправильный порт или путь.

Проверьте код приложения

- Убедитесь, что эндпоинт проверки состояния реализован корректно и действительно отражает готовность/здоровье приложения. Иногда эндпоинт может возвращать успех, даже если приложение сломано.

Ограничения ресурсов

- Недостаток CPU или памяти может привести к тому, что приложение перестанет отвечать, вызывая сбои проб. Проверьте использование ресурсов Pod'a (`kubectl top pod <pod-name>`) и рассмотрите возможность настройки лимитов и запросов ресурсов.

Проблемы с сетью

- В редких случаях политики сети или проблемы с CNI могут препятствовать достижению проб контейнера. Проверьте сетевое соединение внутри кластера.

Рабочие нагрузки

Deployments

- Understanding Deployments
- Creating Deployments
- Managing Deployments
- Troubleshooting by using CLI

DaemonSets

- Понимание DaemonSets
- Создание DaemonSets
- Управление DaemonSets

StatefulSets

- Понимание Sta
- Создание State
- Управление St

Pods

- Понимание Pod'ов
- Пример YAML файла
- Управление Pod с помощью CLI
- Управление Pod через веб-консоль

Контейнер

- Понимание ко
- Понимание эф
- Взаимодейств

Deployments

Содержание

Understanding Deployments

Обратитесь к официальной документации Kubernetes: Deployments ↗

> **Deployment** — это ресурс более высокого уровня в Kubernetes для управления и обновления реплик Pod'ов декларативным способом. Он предоставляет надежный и гибкий способ определения того, как должно работать ваше приложение, включая количество поддерживаемых реплик и безопасное выполнение rolling update.

Deployment — это объект в Kubernetes API, который управляет Pod'ами и ReplicaSet'ами. При создании Deployment Kubernetes автоматически создает ReplicaSet, который отвечает за поддержание указанного количества реплик Pod'ов.

Используя Deployments, вы можете:

- Декларативное управление: определить желаемое состояние приложения, и Kubernetes автоматически обеспечит соответствие фактического состояния кластера желаемому.
- Контроль версий и откат: отслеживать каждую ревизию Deployment и легко откатываться к предыдущей стабильной версии при возникновении проблем.
- Обновления без простоя: постепенно обновлять приложение с помощью стратегии rolling update без прерывания сервиса.

- Самовосстановление: Deployment автоматически заменяет экземпляры Pod'ов при их сбое, завершении или удалении с узла, обеспечивая постоянное наличие заданного количества Pod'ов.

Как это работает:

1. Вы определяете желаемое состояние приложения через Deployment (например, какой образ использовать, сколько реплик запускать).
2. Deployment создает ReplicaSet, чтобы обеспечить запуск указанного количества Pod'ов.
3. ReplicaSet создает и управляет фактическими экземплярами Pod'ов.
4. При обновлении Deployment (например, смене версии образа) создается новый ReplicaSet, который постепенно заменяет старые Pod'ы новыми согласно стратегии rolling update, пока все новые Pod'ы не запустятся, после чего удаляется старый ReplicaSet.

Creating Deployments

Creating a Deployment by using CLI

Prerequisites

- Убедитесь, что `kubectl` настроен и подключен к вашему кластеру.

YAML file example

```yaml
# example-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment # Имя Deployment
  labels:
    app: nginx # Метки для идентификации и выбора
spec:
  replicas: 3 # Желаемое количество реплик Pod
  selector:
    matchLabels:
      app: nginx # Селектор для выбора Pod'ов, управляемых этим Deploymen
t
  template:
    metadata:
      labels:
        app: nginx # Метки Pod, должны совпадать с selector.matchLabels
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2 # Образ контейнера
          ports:
            - containerPort: 80 # Открытый порт контейнера
          resources: # Лимиты и запросы ресурсов
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 200m
              memory: 256Mi
```

Creating a Deployment via YAML

```yaml
# Шаг 1: Создать Deployment из yaml
kubectl apply -f example-deployment.yaml

# Шаг 2: Проверить статус Deployment
kubectl get deployment nginx-deployment # Просмотр Deployment
kubectl get pod -l app=nginx # Просмотр Pod'ов, созданных этим Deployment
```

Creating a Deployment by using web console

Prerequisites

Получите адрес образа. Источником образов могут быть репозитории образов, интегрированные администратором платформы через toolchain, либо сторонние репозитории образов.

- В первом случае администратор обычно назначает репозиторий образов вашему проекту, и вы можете использовать образы из него. Если нужный репозиторий не найден, обратитесь к администратору для выделения.
- Если это сторонний репозиторий, убедитесь, что образы можно напрямую вытягивать из него в текущем кластере.
- Если для реестра образов требуется аутентификация, необходимо настроить соответствующий image pull secret. Подробнее см. Add ImagePullSecrets to ServiceAccount.

Procedure - Configure Basic Info

1. В **Container Platform** перейдите в **Workloads** > **Deployments** в левом меню.
2. Нажмите **Create Deployment**.
3. **Выберите** или **введите** образ и нажмите **Confirm**.

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, можно фильтровать образы по **Already Integrated**. **Integration Project Name** — например, образы (registry-projectname), где projectname — имя проекта в веб-консоли и имя проекта в репозитории образов.

4. В разделе **Basic Info** настройте декларативные параметры для Deployment:

Параметры	Описание
Replicas	Определяет желаемое количество реплик Pod в Deployment (по умолчанию: `1`). Настраивается в зависимости от требований

Параметры	Описание
	нагрузки.
More > **Update Strategy**	Настройка стратегии `rollingUpdate` для обновлений без простоя: **Max surge** (`maxSurge`): • Максимальное количество Pod'ов, превышающих желаемое число реплик во время обновления. • Принимает абсолютные значения (например, `2`) или проценты (например, `20%`). • Расчет процентов: `ceil(current_replicas × percentage)` . • Пример: 4.1 → `5` при расчете от 10 реплик. **Max unavailable** (`maxUnavailable`): • Максимальное количество Pod'ов, которые могут быть временно недоступны во время обновления. • Процентные значения не могут превышать `100%` . • Расчет процентов: `floor(current_replicas × percentage)` . • Пример: 4.9 → `4` при расчете от 10 реплик. **Примечания**: 1. **Значения по умолчанию**: `maxSurge=1` , `maxUnavailable=1` , если явно не заданы. 2. **Незапущенные Pod'ы** (например, в состояниях `Pending` / `CrashLoopBackOff`) считаются недоступными. 3. **Одновременные ограничения**: • `maxSurge` и `maxUnavailable` не могут одновременно быть `0` или `0%` . • Если процентные значения обоих параметров равны `0` , Kubernetes принудительно устанавливает `maxUnavailable=1` для обеспечения прогресса обновления.

Параметры	Описание
	Пример: Для Deployment с 10 репликами: • `maxSurge=2` → Общее количество Pod'ов во время обновления: `10 + 2 = 12`. • `maxUnavailable=3` → Минимальное количество доступных Pod'ов: `10 - 3 = 7`. • Это обеспечивает доступность при контролируемом развертывании.

Procedure - Configure Pod

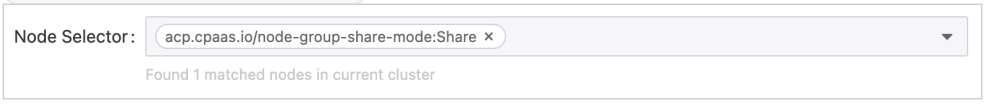
Примечание: В кластерах с разной архитектурой, при развертывании образов для одной архитектуры, убедитесь в правильной настройке Node Affinity Rules для планирования Pod.

1. В разделе **Pod** настройте параметры контейнерного рантайма и управления жизненным циклом:

Параметры	Описание
Volumes	Монтирование постоянных томов в контейнеры. Поддерживаемые типы томов: `PVC`, `ConfigMap`, `Secret`, `emptyDir`, `hostPath` и др. Для деталей реализации см. Volume Mounting Guide.
Pull Secret	Требуется **только** при вытягивании образов из сторонних реестров (при ручном вводе URL образа). **Примечание**: Секрет для аутентификации при вытягивании из защищенного реестра.
Close Grace Period	Время (по умолчанию: `30s`), отведенное Pod для корректного завершения после получения сигнала завершения. - В этот период Pod завершает текущие запросы и освобождает ресурсы.

Параметры	Описание
	- Установка `0` приводит к немедленному удалению (SIGKILL), что может вызвать прерывание запросов.

2. **Node Affinity Rules**

Параметры	Описание
More > **Node Selector**	Ограничение Pod'ов узлами с определенными метками (например, `kubernetes.io/os: linux`). 
More > **Affinity**	Определение тонких правил планирования на основе существующих. **Типы Affinity**: • **Pod Affinity**: Планирование новых Pod'ов на узлы, где уже размещены определённые Pod'ы (одинаковый топологический домен). • **Pod Anti-affinity**: Запрет совместного размещения новых Pod'ов с определёнными Pod'ами. **Режимы применения**: • `requiredDuringSchedulingIgnoredDuringExecution`: Pod'ы планируются *только* если правила соблюдены. • `preferredDuringSchedulingIgnoredDuringExecution`: Приоритет узлам, удовлетворяющим правилам, но допускаются исключения. **Поля конфигурации**: • `topologyKey`: Метка узла, определяющая топологический домен (по умолчанию: `kubernetes.io/hostname`). • `labelSelector`: Фильтр целевых Pod'ов по меткам.

3. **Network Configuration**

- Kube-OVN

Параметры	Описание
Bandwidth Limits	Ограничение QoS для сетевого трафика Pod: • **Ограничение исходящего трафика**: Максимальная скорость исходящего трафика (например, `10Mbps`). • **Ограничение входящего трафика**: Максимальная скорость входящего трафика.
Subnet	Назначение IP из предопределенного пула подсетей. Если не указано, используется подсеть по умолчанию для namespace.
Static IP Address	Привязка постоянных IP-адресов к Pod'ам: • Несколько Pod'ов из разных Deployments могут запрашивать один IP, но одновременно использовать его может только один Pod. • **Критично**: Количество статических IP должно быть ≥ количеству реплик Pod.

- Calico

Параметры	Описание
Static IP Address	Назначение фиксированных IP с строгой уникальностью: • Каждый IP может быть привязан **только к одному Pod** в кластере. • **Критично**: Количество статических IP должно быть ≥ количеству реплик Pod.

Procedure - Configure Containers

1. В разделе **Container** настройте соответствующую информацию согласно следующим инструкциям.

Параметры	Описание
Resource Requests & Limits	• **Requests**: Минимальные CPU/память, необходимые для работы контейнера. • **Limits**: Максимальные CPU/память, разрешённые во время выполнения контейнера. Для определения единиц см. Resource Units. **Коэффициент overcommit в namespace**: • **Без overcommit**: Если в namespace заданы квоты ресурсов: Requests/limits контейнера наследуют значения по умолчанию из namespace (можно изменить). Без квот в namespace: Нет значений по умолчанию; задается кастомный Request. • **С overcommit**: Requests рассчитываются автоматически как `Limits / Overcommit ratio` (неизменяемо). **Ограничения**: • Request ≤ Limit ≤ максимальная квота namespace. • Изменение коэффициента overcommit требует пересоздания pod для вступления в силу. • При overcommit отключается ручная настройка Request. • Отсутствие квот в namespace → отсутствие ограничений ресурсов контейнера.
Extended Resources	Настройка расширенных ресурсов, доступных в кластере (например, vGPU, pGPU).
Volume Mounts	Конфигурация постоянного хранилища. См. Storage Volume Mounting Instructions. **Операции**:

Параметры	Описание
	• Для существующих томов pod: нажмите **Add** • Если томов pod нет: нажмите **Add & Mount** **Параметры**: • `mountPath`: Путь в файловой системе контейнера (например, `/data`) • `subPath`: Относительный путь файла/директории внутри тома. Для `ConfigMap` / `Secret`: выбор конкретного ключа • `readOnly`: Монтировать только для чтения (по умолчанию — чтение-запись) См. Kubernetes Volumes↗.
Ports	Открытие портов контейнера. **Пример**: Открыть TCP порт `6379` с именем `redis`. **Поля**: • `protocol`: TCP/UDP • `Port`: Открываемый порт (например, `6379`) • `name`: DNS-совместимый идентификатор (например, `redis`)
Startup Commands & Arguments	Переопределение стандартных ENTRYPOINT/CMD: **Пример 1**: Выполнить `top -b` - **Command**: `["top", "-b"]` - **ИЛИ** Command: `["top"]`, Args: `["-b"]` **Пример 2**: Выводить `$MESSAGE`: `/bin/sh -c "while true; do echo $(MESSAGE); sleep 10; done"` См. Defining Commands↗.

Параметры	Описание
More > **Environment Variables**	• Статические значения: прямые пары ключ-значение • Динамические значения: ссылки на ключи ConfigMap/Secret, поля pod (`fieldRef`), метрики ресурсов (`resourceFieldRef`) **Примечание**: Переменные окружения переопределяют настройки образа/конфигурационных файлов.
More > **Referenced ConfigMaps**	Внедрение полного ConfigMap/Secret как переменных окружения. Поддерживаемые типы Secret: `Opaque` , `kubernetes.io/basic-auth` .
More > **Health Checks**	• **Liveness Probe**: Проверка здоровья контейнера (перезапуск при сбое) • **Readiness Probe**: Проверка доступности сервиса (удаление из endpoints при сбое) См. Health Check Parameters.
More > **Log Files**	Настройка путей логов: - По умолчанию: сбор stdout - Шаблоны файлов: например, `/var/log/*.log` **Требования**: • Драйвер хранения `overlay2` : поддерживается по умолчанию • `devicemapper` : необходимо вручную монтировать EmptyDir в каталог логов • Узлы Windows: убедитесь, что родительский каталог смонтирован (например, `c:/a` для `c:/a/b/c/*.log`)
More > **Exclude Log Files**	Исключение определённых логов из сбора (например, `/var/log/aaa.log`).

Параметры	Описание
More > **Execute before Stopping**	Выполнение команд перед завершением контейнера. **Пример**: `echo "stop"` **Примечание**: Время выполнения команды должно быть меньше `terminationGracePeriodSeconds` pod.

2. Нажмите **Add Container** (вверху справа) ИЛИ **Add Init Container**.
   См. Init Containers ↗. Init Container:
   1. Запускается перед основными контейнерами приложения (последовательное выполнение).
   2. Освобождает ресурсы после завершения.
   3. Удаление разрешено, если:
      - В Pod >1 контейнера приложения И ≥1 init контейнер.
      - Не разрешено для Pod с одним контейнером приложения.
3. Нажмите **Create**.

Reference Information

Storage Volume Mounting instructions

Тип	Назначение
Persistent Volume Claim	Привязка существующего PVC для запроса постоянного хранилища. **Примечание**: Выбираются только привязанные PVC (с ассоциированным PV). Непривязанные PVC вызовут ошибку создания pod.

Тип	Назначение
ConfigMap	Монтирование полного/частичного содержимого ConfigMap как файлов: • Полный ConfigMap: создаёт файлы с именами ключей под путём монтирования • Выбор подпути: монтирование конкретного ключа (например, `my.cnf`)
Secret	Монтирование полного/частичного содержимого Secret как файлов: • Полный Secret: создаёт файлы с именами ключей под путём монтирования • Выбор подпути: монтирование конкретного ключа (например, `tls.crt`)
Ephemeral Volumes	Временный том, предоставляемый кластером, с возможностями: • Динамическое выделение • Жизненный цикл связан с pod • Поддержка декларативной конфигурации **Сценарий использования**: временное хранение данных. См. Ephemeral Volumes
Empty Directory	Временное хранилище для совместного использования между контейнерами в одном pod: • Создаётся на узле при старте pod • Удаляется при удалении pod **Сценарий использования**: обмен файлами между контейнерами, временное хранение данных. См. EmptyDir
Host Path	Монтирование директории хост-машины (должна начинаться с `/`, например, `/volumepath`).

Heath Checks

- Пример YAML файла для health checks
- Параметры настройки health checks в веб-консоли

Managing Deployments

Managing a Deployment by using CLI

Viewing a Deployment

- Проверьте, что Deployment создан.

```
kubectl get deployments
```

- Получите подробности о вашем Deployment.

```
kubectl describe deployments
```

Updating a Deployment

Выполните следующие шаги для обновления Deployment:

1. Обновим Pod'ы nginx, чтобы использовать образ nginx:1 .16.1.

   ```
   kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.
   1
   ```

   или используйте команду:

   ```
   kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1
   ```

   Также можно отредактировать Deployment и изменить `.spec.template.spec.containers[0].image` с `nginx:1.14.2` на `nginx:1.16.1`:

```
kubectl edit deployment/nginx-deployment
```

2. Чтобы увидеть статус развертывания, выполните:

```
kubectl rollout status deployment/nginx-deployment
```

Выполните `kubectl get rs`, чтобы увидеть, что Deployment обновил Pod'ы, создав новый ReplicaSet и масштабируя его до 3 реплик, а также уменьшил старый ReplicaSet до 0 реплик.

```
kubectl get rs
```

Выполнение `kubectl get pods` теперь должно показывать только новые Pod'ы:

```
kubectl get pods
```

Scaling a Deployment

Вы можете масштабировать Deployment с помощью следующей команды:

```
kubectl scale deployment/nginx-deployment --replicas=10
```

Rolling Back a Deployment

- Предположим, что при обновлении Deployment вы допустили опечатку в имени образа, указав `nginx:1.161` вместо `nginx:1.16.1`:

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.161
```

- Развертывание застряло. Вы можете проверить это, посмотрев статус развертывания:

```
kubectl rollout status deployment/nginx-deployment
```

Deleting a Deployment

Удаление Deployment также удалит управляемый им ReplicaSet и все связанные Pod'ы.

```
kubectl delete deployment <deployment-name>
```

Managing a Deployment by using web console

Viewing a Deployment

Вы можете просмотреть Deployment, чтобы получить информацию о вашем приложении.

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. Нажмите на имя Deployment, чтобы увидеть **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Updating a Deployment

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. В выпадающем меню **Actions** выберите **Update**, чтобы открыть страницу редактирования Deployment.

Deleting a Deployment

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. В выпадающем меню **Actions** нажмите кнопку **Delete** в столбце операций и подтвердите удаление.

Troubleshooting by using CLI

Если у Deployment возникают проблемы, ниже приведены распространённые методы диагностики.

Check Deployment status

```
kubectl get deployment nginx-deployment
kubectl describe deployment nginx-deployment # Просмотр подробных событий и статуса
```

Check ReplicaSet status

```
kubectl get rs -l app=nginx
kubectl describe rs <replicaset-name>
```

Check Pod status

```
kubectl get pods -l app=nginx
kubectl describe pod <pod-name>
```

View Logs

```
kubectl logs <pod-name> -c <container-name> # Просмотр логов конкретного контейнера
kubectl logs <pod-name> --previous          # Просмотр логов ранее завершённого контейнера
```

Enter Pod for debugging

```
kubectl exec -it <pod-name> -- /bin/bash # Вход в shell контейнера
```

Check Health configuration

Убедитесь, что livenessProbe и readinessProbe настроены корректно, а эндпоинты проверки здоровья вашего приложения отвечают должным образом. Troubleshooting probe failures

Check Resource Limits

Убедитесь, что запросы и лимиты ресурсов контейнеров разумны и контейнеры не завершаются из-за нехватки ресурсов.

DaemonSets

Содержание

Понимание DaemonSets

Обратитесь к официальной документации Kubernetes: DaemonSets ↗

DaemonSet — это контроллер Kubernetes, который гарантирует, что на всех (или на подмножестве) узлов кластера запущена ровно одна копия указанного Pod. В отличие от Deployments, DaemonSets ориентированы на узлы, а не на приложения, что делает их идеальными для развертывания инфраструктурных сервисов по всему кластеру, таких как сборщики логов, агенты мониторинга или демоны хранения.

WARNING

Операционные заметки по DaemonSet

1. **Характеристики поведения**
   - **Распределение Pod**: DaemonSet разворачивает ровно одну **копию Pod** на каждый планируемый **Node**, соответствующий его критериям:
     - Разворачивает ровно **одну копию Pod на каждый планируемый узел**, который:
       - Соответствует критериям `nodeSelector` или `nodeAffinity` (если указаны).
       - Не находится в состоянии `NotReady`.
       - Не имеет **Taints** `NoSchedule` или `NoExecute`, если только в **Pod Template** не настроены соответствующие **Tolerations**.
   - **Формула количества Pod**: **Количество Pod**, управляемых DaemonSet, **равно количеству подходящих узлов**.
   - **Обработка узлов с двойной ролью**: Узлы, выполняющие одновременно роли **Control Plane** и **Worker Node**, будут запускать только одну копию **Pod** DaemonSet, независимо от их меток ролей, при условии, что они планируемы.
2. **Ключевые ограничения** (исключённые узлы)
   - Узлы, явно помеченные как `Unschedulable: true` (например, с помощью `kubectl cordon`).

- Узлы со статусом `NotReady`.
- Узлы с несовместимыми **Taints**, для которых в **Pod Template** DaemonSet не настроены соответствующие Tolerations.

Создание DaemonSets

Создание DaemonSet с помощью CLI

Предварительные требования

- Убедитесь, что `kubectl` настроен и подключён к вашему кластеру.

Пример YAML файла

```yaml
# example-daemonSet.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    k8s-app: fluentd-logging
spec:
  selector: # определяет, как DaemonSet идентифицирует управляемые Pod. Должен совпадать с `template.metadata.label`s.
    matchLabels:
      name: fluentd-elasticsearch
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template: # определяет Pod Template для DaemonSet. Каждый Pod, созданный этим DaemonSet, будет соответствовать этому шаблону
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations: # эти tolerations позволяют запускать daemonset на узлах control plane, удалите их, если узлы control plane не должны запускать Pod
        - key: node-role.kubernetes.io/control-plane
          operator: Exists
          effect: NoSchedule
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
      containers:
        - name: fluentd-elasticsearch
          image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
          resources:
            limits:
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          volumeMounts:
            - name: varlog
              mountPath: /var/log
```

```
      # возможно, стоит задать высокий priorityClass, чтобы Pod DaemonSet
      # мог вытеснять запущенные Pod
      # priorityClassName: important
      terminationGracePeriodSeconds: 30
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
```

Создание DaemonSet через YAML

```
# Шаг 1: Для создания DaemonSet, определённого в *example-daemonSet.yaml
*, выполните команду
kubectl apply -f example-daemonSet.yaml

# Шаг 2: Для проверки создания и статуса DaemonSet и связанных с ним Pod:
kubectl get daemonset fluentd-elasticsearch # Просмотр DaemonSet
kubectl get pods -l name=fluentd-elasticsearch -o wide # Проверка Pod, уп
равляемых этим DaemonSet, на конкретных узлах
```

Создание DaemonSet через веб-консоль

Предварительные требования

Получите адрес образа. Источником образов может быть репозиторий образов, интегрированный администратором платформы через toolchain, либо репозитории образов сторонних платформ.

- В первом случае администратор обычно назначает репозиторий образов вашему проекту, и вы можете использовать образы из него. Если нужный репозиторий образов не найден, обратитесь к администратору для выделения.
- Если это репозиторий образов сторонней платформы, убедитесь, что образы можно напрямую загрузить из него в текущем кластере.
- Если для реестра образов требуется аутентификация, необходимо настроить соответствующий image pull secret. Подробнее см. Add ImagePullSecrets to ServiceAccount.

Процедура — Настройка базовой информации

1. В **Container Platform** перейдите в раздел **Workloads** > **DaemonSets** в левой боковой панели.
2. Нажмите **Create DaemonSet**.
3. **Выберите** или **введите** образ и нажмите **Confirm**.

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, можно фильтровать образы по категории **Already Integrated**. **Integration Project Name**, например, образы (registry-projectname), включает имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

В разделе **Basic Info** настройте декларативные параметры для DaemonSet:

Параметры	Описание
More > **Update Strategy**	Настраивает стратегию `rollingUpdate` для обновления DaemonSet Pod без простоя. **Max unavailable** (`maxUnavailable`): Максимальное количество Pod, которые могут быть временно недоступны во время обновления. Принимает абсолютные значения (например, 1) или проценты (например, 10%). **Пример**: Если узлов 10, а maxUnavailable равен 10%, то floor(10 * 0.1) = 1 Pod может быть недоступен. **Примечания:** • **Значения по умолчанию**: Если явно не указано, `maxSurge` по умолчанию 0, а `maxUnavailable` — 1 (или 10%, если указано в процентах). • **Незапущенные Pod**: Pod в состояниях `Pending` или `CrashLoopBackOff` считаются недоступными. • **Одновременные ограничения**: `maxSurge` и `maxUnavailable` не могут одновременно быть 0 или 0%. Если процентные значения приводят к 0 для обоих параметров, Kubernetes принудительно устанавливает `maxUnavailable=1` для обеспечения прогресса обновления.

Процедура — Настройка Pod

Раздел **Pod**, см. Deployment - Configure Pod

Процедура — Настройка контейнеров

Раздел **Containers**, см. Deployment - Configure Containers

Процедура — Создание

Нажмите **Create**.

После нажатия **Create** DaemonSet:

- ✅ Автоматически развернёт копии Pod на всех подходящих узлах, соответствующих:
  - критериям `nodeSelector` (если определены).
  - настройкам `tolerations` (позволяющим планировать Pod на узлах с taints).
  - узлам в состоянии `Ready` и с `Schedulable: true`.
- ❌ Исключённые узлы:
  - Узлы с taint `NoSchedule` (если явно не допускается).
  - Узлы, вручную заблокированные (`kubectl cordon`).
  - Узлы в состояниях `NotReady` или `Unschedulable`.

Управление DaemonSets

Управление DaemonSet с помощью CLI

Просмотр DaemonSet

- Получить сводку всех DaemonSet в namespace:

```
kubectl get daemonsets -n <namespace>
```

- Получить подробную информацию о конкретном DaemonSet, включая события и статус Pod:

```
kubectl describe daemonset <daemonset-name>
```

Обновление DaemonSet

При изменении **Pod Template** DaemonSet (например, смена образа контейнера или добавление volume mount) Kubernetes по умолчанию выполняет rolling update (если `updateStrategy.type` установлен в `RollingUpdate`, что является значением по умолчанию).

- Сначала отредактируйте YAML файл (например, `example-daemonset.yaml`) с нужными изменениями, затем примените его:

```
kubectl apply -f example-daemonset.yaml
```

- Можно отслеживать прогресс rolling update:

```
kubectl rollout status daemonset/<daemonset-name>
```

Удаление DaemonSet

Для удаления DaemonSet и всех управляемых им Pod:

```
kubectl delete daemonset <daemonset-name>
```

Управление DaemonSet через веб-консоль

Просмотр DaemonSet

1. В **Container Platform** перейдите в **Workloads** > **DaemonSets**.
2. Найдите нужный DaemonSet.
3. Нажмите на имя DaemonSet для просмотра **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Обновление DaemonSet

1. В **Container Platform** перейдите в **Workloads** > **DaemonSets**.
2. Найдите DaemonSet для обновления.
3. В выпадающем меню **Actions** выберите **Update**, чтобы открыть страницу редактирования DaemonSet, где можно изменить `Replicas`, `image`, `updateStrategy` и др.

Удаление DaemonSet

1. В **Container Platform** перейдите в **Workloads** > **DaemonSets**.

2. Найдите DaemonSet для удаления.
3. В выпадающем меню **Actions** нажмите кнопку **Delete** в колонке операций и подтвердите.

StatefulSets

Содержание

Понимание StatefulSets

Обратитесь к официальной документации Kubernetes: StatefulSets ↗

StatefulSet — это объект API рабочих нагрузок Kubernetes, предназначенный для управления stateful-приложениями, предоставляя:

- **Стабильную сетевую идентичность**: DNS-имя хоста `<statefulset-name>-<ordinal>.<service-name>.ns.svc.cluster.local`.
- **Стабильное постоянное хранилище**: через `volumeClaimTemplates`.
- **Упорядоченное развертывание/масштабирование**: последовательное создание/удаление Pod-ов: Pod-0 → Pod-1 → Pod-N.
- **Упорядоченные rolling updates**: обновления Pod-ов в обратном порядке: Pod-N → Pod-0.

В распределённых системах несколько StatefulSets могут быть развернуты как отдельные компоненты для предоставления специализированных stateful-сервисов (например, *Kafka brokers*, *MongoDB shards*).

Создание StatefulSets

Создание StatefulSet с помощью CLI

Предварительные требования

- Убедитесь, что `kubectl` настроен и подключён к вашему кластеру.

Пример YAML-файла

```yaml
# example-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # должен совпадать с .spec.template.metadata.labels
  serviceName: 'nginx' # этот headless Service отвечает за сетевую иденти
чность Pod-ов
  replicas: 3 # задаёт желаемое количество реплик Pod-ов (по умолчанию:
1)
  minReadySeconds: 10 # по умолчанию 0
  template: # шаблон Pod-а для StatefulSet
    metadata:
      labels:
        app: nginx # должен совпадать с .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: nginx
          image: registry.k8s.io/nginx-slim:0.24
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
  volumeClaimTemplates: # шаблоны PersistentVolumeClaim (PVC). Каждый Pod
получает уникальный PersistentVolume (PV), динамически выделенный на осно
ве этих шаблонов.
    - metadata:
        name: www
      spec:
        accessModes: ['ReadWriteOnce']
        storageClassName: 'my-storage-class'
        resources:
          requests:
            storage: 1Gi
---
# example-service.yaml
apiVersion: v1
```

```
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
    - port: 80
      name: web
  clusterIP: None
  selector:
    app: nginx
```

Создание StatefulSet через YAML

```
# Шаг 1: Для создания StatefulSet, определённого в *example-statefulset.yaml*, выполните команду
kubectl apply -f example-statefulset.yaml

# Шаг 2: Для проверки создания и статуса StatefulSet, а также связанных Pod-ов и PVC:
kubectl get statefulset web # Просмотр StatefulSet
kubectl get pods -l app=nginx # Проверка Pod-ов, управляемых этим StatefulSet
kubectl get pvc -l app=nginx # Проверка PVC, созданных volumeClaimTemplates
```

Создание StatefulSet через веб-консоль

Предварительные требования

Получите адрес образа. Источником образов могут быть репозитории образов, интегрированные администратором платформы через toolchain, либо репозитории образов сторонних платформ.

- В первом случае администратор обычно назначает репозиторий образов вашему проекту, и вы можете использовать образы из него. Если нужный репозиторий образов не найден, обратитесь к администратору для выделения.

- Если это репозиторий образов сторонней платформы, убедитесь, что образы можно напрямую подтягивать из него в текущем кластере.
- Если для реестра образов требуется аутентификация, необходимо настроить соответствующий image pull secret. Подробнее см. Add ImagePullSecrets to ServiceAccount.

Процедура — Настройка базовой информации

1. В **Container Platform** перейдите в **Workloads** > **StatefulSets** в левой боковой панели.
2. Нажмите **Create StatefulSet**.
3. **Выберите** или **введите** образ и нажмите **Confirm**.

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, можно фильтровать образы по **Already Integrated**. **Integration Project Name**, например, образы (registry-projectname), включает имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

В разделе **Basic Info** настройте декларативные параметры для рабочих нагрузок StatefulSet:

Параметры	Описание
Replicas	Определяет желаемое количество реплик Pod-ов в StatefulSet (по умолчанию: 1). Настраивается в зависимости от требований к нагрузке и ожидаемого объёма запросов.
Update Strategy	Управляет поэтапными обновлениями во время rolling update StatefulSet. Стратегия `RollingUpdate` является значением по умолчанию и рекомендуется. **Partition**: порог по индексу для обновления Pod-ов. • Pod-ы с индексом ≥ `partition` обновляются сразу. • Pod-ы с индексом < `partition` сохраняют предыдущую спецификацию. **Пример**:

Параметры	Описание
	• `Replicas=5` (Pod-ы: web-0 ~ web-4) • `Partition=3` (обновляются только web-3 и web-4)

Volume Claim Templates	`volumeClaimTemplates` — ключевая функция StatefulSets, позволяющая динамически выделять постоянное хранилище для каждого Pod-а. Каждая реплика Pod-а в StatefulSet автоматически получает собственный уникальный PersistentVolumeClaim (PVC) на основе предопределённых шаблонов.
	• 1. **Динамическое создание PVC**: автоматически создаёт уникальные PVC для каждого Pod-а с шаблоном имени: `<statefulset-name>-<claim-template-name>-<pod-ordinal>`. **Пример**: `web-www-web-0`, `web-www-web-1`.
	• 2. **Режимы доступа**: поддерживает все режимы доступа Kubernetes.
	  • ReadWriteOnce (RWO — однопользовательский режим чтения/записи)
	  • ReadOnlyMany (ROX — многопользовательский режим только для чтения)
	  • ReadWriteMany (RWX — многопользовательский режим чтения/записи).
	• 3. **Storage Class**: указывается backend хранилища через storageClassName. Если не указано, используется StorageClass по умолчанию в кластере. Поддерживает различные типы хранилищ (например, SSD, HDD) в облаке или локально.
	• 4. **Ёмкость**: настраивается через resources.requests.storage. **Пример**: 1Gi. Поддерживается динамическое расширение томов, если это разрешено StorageClass.

Процедура — Настройка Pod-а

Раздел **Pod**, см. Deployment - Configure Pod

Процедура — Настройка контейнеров

Раздел **Containers**, см. Deployment - Configure Containers

Процедура — Создание

Нажмите **Create**.

Проверка состояния (Health Checks)

- Пример YAML файла для health checks
- Параметры настройки health checks в веб-консоли

Управление StatefulSets

Управление StatefulSet с помощью CLI

Просмотр StatefulSet

Вы можете просмотреть StatefulSet, чтобы получить информацию о вашем приложении.

- Проверьте, что StatefulSet создан.

```
kubectl get statefulsets
```

- Получите подробности о вашем StatefulSet.

```
kubectl describe statefulsets
```

Масштабирование StatefulSet

- Чтобы изменить количество реплик для существующего StatefulSet:

```
kubectl scale statefulset <statefulset-name> --replicas=<new-replica-co
unt>
```

- Пример:

```
kubectl scale statefulset web --replicas=5
```

Обновление StatefulSet (Rolling Update)

При изменении шаблона Pod-а StatefulSet (например, смена образа контейнера) Kubernetes по умолчанию выполняет rolling update (если updateStrategy установлен в RollingUpdate, что является значением по умолчанию).

- Сначала отредактируйте YAML-файл (например, example-statefulset.yaml) с нужными изменениями, затем примените его:

```
kubectl apply -f example-statefulset.yaml
```

- Затем вы можете отслеживать прогресс rolling update:

```
kubectl rollout status statefulset/<statefulset-name>
```

Удаление StatefulSet

Чтобы удалить StatefulSet и связанные с ним Pod-ы:

```
kubectl delete statefulset <statefulset-name>
```

По умолчанию удаление StatefulSet не удаляет связанные PersistentVolumeClaims (PVC) или PersistentVolumes (PV), чтобы избежать потери данных. Чтобы также удалить PVC, сделайте это явно:

```
kubectl delete pvc -l app=<label-selector-for-your-statefulset> # Пример:
kubectl delete pvc -l app=nginx
```

Альтернативно, если ваши `volumeClaimTemplates` используют `StorageClass` с `reclaimPolicy` равным `Delete`, PV и подлежащие хранилища будут удалены автоматически при удалении PVC.

Управление StatefulSet через веб-консоль

Просмотр StatefulSet

1. В **Container Platform** перейдите в **Workloads** > **StatefulSets**.
2. Найдите StatefulSet, который хотите просмотреть.
3. Нажмите на имя StatefulSet, чтобы увидеть **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Обновление StatefulSet

1. В **Container Platform** перейдите в **Workloads** > **StatefulSets**.
2. Найдите StatefulSet, который хотите обновить.
3. В выпадающем меню **Actions** выберите **Update**, чтобы открыть страницу редактирования StatefulSet, где можно обновить `Replicas`, `image`, `updateStrategy` и др.

Удаление StatefulSet

1. В **Container Platform** перейдите в **Workloads** > **StatefulSets**.
2. Найдите StatefulSet, который хотите удалить.
3. В выпадающем меню **Actions** нажмите кнопку **Delete** в колонке операций и подтвердите.

CronJobs

Содержание

Понимание CronJobs

Обратитесь к официальной документации Kubernetes:

- CronJobs ↗
- Запуск автоматизированных задач с помощью CronJob ↗

CronJob определяет задачи, которые выполняются до завершения и затем останавливаются. Они позволяют запускать один и тот же Job несколько раз в соответствии с расписанием.

CronJob — это тип контроллера рабочих нагрузок в Kubernetes. Вы можете создать CronJob через веб-консоль или CLI для периодического или повторяющегося запуска непостоянной программы, такой как запланированные резервные копии, запланированная очистка или запланированная отправка электронной почты.

Создание CronJobs

Создание CronJob с помощью CLI

Предварительные требования

- Убедитесь, что `kubectl` настроен и подключен к вашему кластеру.

Пример YAML-файла

```yaml
# example-cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Создание CronJob через YAML

```
kubectl apply -f example-cronjob.yaml
```

Создание CronJobs с помощью веб-консоли

Предварительные требования

Получите адрес образа. Образы могут быть взяты из реестра образов, интегрированного администратором платформы через toolchain, или из сторонних реестров образов.

- Для образов из интегрированного реестра администратор обычно назначает реестр образов вашему проекту, что позволяет использовать образы внутри него. Если нужный реестр образов не найден, обратитесь к администратору для выделения.

- Если используется сторонний реестр образов, убедитесь, что образы можно напрямую загрузить из него в текущем кластере.
- Если реестр образов требует аутентификации, необходимо настроить соответствующий секрет для загрузки образов. Подробнее см. Добавление ImagePullSecrets к ServiceAccount.

Процедура — Настройка базовой информации

1. В **Container Platform** перейдите в **Workloads** > **CronJobs** в левой боковой панели.
2. Нажмите **Create CronJob**.
3. **Выберите** или **введите** образ и нажмите **Confirm**.
   Примечание: Фильтрация образов доступна только при использовании образов из интегрированного реестра платформы. Например, интегрированный проект с именем containers (registry-projectname) означает, что projectname — имя проекта платформы, а containers — имя проекта реестра образов.
4. В разделе **Cron Configuration** настройте способ выполнения задачи и связанные параметры.
   Execute Type:
   - **Manual**: Ручное выполнение требует явного запуска задачи вручную для каждого запуска.
   - **Scheduled**: Запланированное выполнение требует настройки следующих параметров расписания:

Параметр	Описание
Schedule	Определяет расписание cron с использованием синтаксиса Crontab ↗. Контроллер CronJob рассчитывает следующее время выполнения с учетом выбранного часового пояса. **Примечания**: • Для Kubernetes версий < v1.25: выбор часового пояса не поддерживается; расписание ДОЛЖНО использовать UTC. • Для Kubernetes версий ≥ v1.25: поддерживается расписание с учетом часового пояса (по умолчанию — локальный часовой

Параметр	Описание
	пояс пользователя).
Concurrency Policy	Указывает, как обрабатываются параллельные выполнения Job (`Allow`, `Forbid` или `Replace` согласно спецификации K8s↗).

Сохранение истории Job:

- Установите лимиты хранения для завершенных Job:
  - **History Limits**: лимит истории успешных задач (по умолчанию: 20)
  - **Failed Jobs**: лимит истории неудачных задач** (по умолчанию: 20)
- При превышении лимитов хранения сначала удаляются самые старые задачи.

5. В разделе **Job Configuration** выберите тип Job. CronJob управляет Job, состоящими из Pod. Настройте шаблон Job в зависимости от типа вашей рабочей нагрузки:

Параметр	Описание
Job Type	Выберите режим завершения Job (`Non-parallel`, `Parallel with fixed completion count` или `Indexed Job` согласно паттернам Job в K8s↗).
Backoff Limit	Установите максимальное количество попыток повторного запуска перед пометкой Job как неудачной.

Процедура — Настройка Pod

- Раздел **Pod**, см. Deployment - Configure Pod

Процедура — Настройка контейнеров

- Раздел **Container**, см. Deployment - Configure Containers

Создание

- Нажмите **Create**.

Немедленное выполнение

Найти ресурс CronJob

- **веб-консоль**: в **Container Platform** перейдите в **Workloads** > **CronJobs** в левой боковой панели.
- **CLI**:

```
kubectl get cronjobs -n <namespace>
```

Запуск выполнения по требованию

- **веб-консоль**: **Execute Immediately**

  1. Нажмите вертикальное многоточие (⋮) справа в списке cronjob.
  2. Выберите **Execute Immediately**. (Или на странице деталей CronJob нажмите Actions в правом верхнем углу и выберите **Execute Immediately**).

- **CLI**:

```
kubectl create job --from=cronjob/<cronjob-name> <job-name> -n <namespace>
```

Проверка деталей Job:

```
kubectl describe job/<job-name> -n <namespace>
kubectl logs job/<job-name> -n <namespace>
```

Мониторинг статуса выполнения

Статус	Описание
Pending	Job создан, но еще не запланирован к выполнению.

Статус	Описание
Running	Pod(ы) Job активно выполняются.
Succeeded	Все Pod, связанные с Job, успешно завершились (код выхода 0).
Failed	По крайней мере один Pod, связанный с Job, завершился с ошибкой (код выхода не равен 0).

Удаление CronJobs

Удаление CronJobs через веб-консоль

1. В **Container Platform** перейдите в **Workloads** > **CronJobs**.
2. Найдите CronJobs, которые хотите удалить.
3. В выпадающем меню **Actions** нажмите кнопку **Delete** и подтвердите.

Удаление CronJobs через CLI

```
kubectl delete cronjob <cronjob-name>
```

Jobs

Содержание

Понимание Jobs

Обратитесь к официальной документации Kubernetes: Jobs ↗

Job предоставляет различные способы определения задач, которые выполняются до завершения и затем останавливаются. Вы можете использовать Job для определения задачи, которая выполняется до завершения один раз.

- **Атомарная единица выполнения**: Каждый Job управляет одним или несколькими Pod до успешного завершения.
- **Механизм повторных попыток**: Управляется параметром `spec.backoffLimit` (по умолчанию: 6).
- **Отслеживание завершения**: Используйте `spec.completions` для определения необходимого количества успешных выполнений.

Пример YAML файла

```yaml
# example-job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: data-processing-job
spec:
  completions: 1 # Количество необходимых успешных выполнений
  parallelism: 1 # Максимальное количество параллельных Pod
  backoffLimit: 3 # Максимальное количество попыток повторного запуска
  template:
    spec:
      restartPolicy: Never # Политика перезапуска для Job (Never/OnFailur
e)
      containers:
        - name: processor
          image: alpine:3.14
          command: ['/bin/sh', '-c']
          args:
            - echo "Processing data..."; sleep 30; echo "Job completed"
```

Обзор выполнения

Каждое выполнение Job в Kubernetes создает отдельный объект Job, что позволяет пользователям:

- **Создавать job с помощью**

```
kubectl apply -f example-job.yaml
```

- **Отслеживать жизненный цикл job с помощью**

```
kubectl get jobs
```

- **Просматривать детали выполнения с помощью**

```
kubectl describe job/<job-name>
```

- **Просматривать логи Pod с помощью**

```
kubectl logs <pod-name>
```

Pods

Содержание

Понимание Pod'ов

Обратитесь к официальной документации Kubernetes на сайте: Pod ↗

Pod — это наименьшая единица вычислений, которую вы можете создать и управлять в Kubernetes. **Pod** (как стая китов или стручок гороха) — это группа из одного или нескольких контейнеров (например, контейнеров), с общим хранилищем и сетевыми ресурсами, а также спецификацией того, как запускать контейнеры. **Pods** являются фундаментальными строительными блоками, на основе которых строятся все контроллеры более высокого уровня (такие как **Deployments**, **StatefulSets**, **DaemonSets**).

Пример YAML файла

pod-example.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx:latest # The container image to use.
      ports:
        - containerPort: 80 # Container ports exposed.
      resources: # Defines CPU and memory requests and limits for the con
tainer.
        requests:
          cpu: '100m'
          memory: '128Mi'
        limits:
          cpu: '200m'
          memory: '256Mi'
```

Управление Pod с помощью CLI

Хотя Pods часто управляются контроллерами более высокого уровня, прямые операции kubectl с Pod полезны для устранения неполадок, инспекции и выполнения разовых

задач.

Просмотр Pod

- Чтобы вывести список всех Pod в текущем namespace:

```
kubectl get pods
```

- Чтобы вывести список всех Pod во всех namespace:

```
kubectl get pods --all-namespaces
# Или короткая версия:
kubectl get pods -A
```

- Чтобы получить подробную информацию о конкретном Pod:

```
kubectl describe pod <pod-name> -n <namespace>

# Пример
kubectl describe pod my-nginx-pod -n default
```

Просмотр логов Pod

- Чтобы просмотреть логи контейнера внутри Pod (полезно для отладки):

```
kubectl logs <pod-name> -n <namespace>
```

- Если в Pod несколько контейнеров, необходимо указать имя контейнера:

```
kubectl logs <pod-name> -c <container-name> -n <namespace>
```

- Чтобы следить за логами в режиме реального времени (поток новых логов по мере их появления):

```
kubectl logs -f <pod-name> -n <namespace>
```

Выполнение команд в Pod

Чтобы выполнить команду внутри конкретного контейнера в Pod (полезно для отладки, например, для доступа к shell):

```
kubectl exec -it <pod-name> -n <namespace> -- <command>

# Пример (для получения shell):
kubectl exec -it my-nginx-pod -n default -- /bin/bash
```

Проброс портов к Pod

Чтобы пробросить локальный порт на порт Pod, позволяя прямой доступ к сервису, работающему внутри Pod с вашей локальной машины (полезно для тестирования или прямого доступа без внешнего экспонирования сервиса):

```
kubectl port-forward <pod-name> <local-port>:<pod-port> -n <namespace>

#Пример
kubectl port-forward my-nginx-pod 8080:80 -n default
```

После выполнения этой команды вы сможете получить доступ к веб-серверу Nginx, работающему в my-nginx-pod, перейдя в браузере по адресу localhost:8080 .

Удаление Pod

- Чтобы удалить конкретный Pod:

  ```
  kubectl delete pod <pod-name> -n <namespace>

  # Пример
  kubectl delete pod my-nginx-pod -n default
  ```

- Чтобы удалить несколько Pod по именам:

```
kubectl delete pod <pod-name-1> <pod-name-2> -n <namespace>
```

- Чтобы удалить Pod по селектору меток (например, удалить все Pod с меткой app=nginx):

```
kubectl delete pods -l app=nginx -n <namespace>
```

Управление Pod через веб-консоль

Просмотр Pod

Интерфейс платформы предоставляет различную информацию о Pod для быстрого ознакомления.

Процедура

1. В **Container Platform** перейдите в **Workloads** > **Pods** в левой боковой панели.
2. Найдите Pod, который хотите просмотреть.
3. Нажмите на имя деплоя, чтобы увидеть **Details**, **YAML**, **Configuration**, **Logs**, **Events**, **Monitoring** и т.д.

Параметры Pod

Ниже приведены пояснения к некоторым параметрам:

Параметр	Описание
Resource Requests & Limits	**Resource Requests** и **Limits** определяют границы потребления CPU и памяти для контейнеров внутри Pod, которые затем суммируются для формирования общего профиля ресурсов Pod. Эти значения критически важны для планировщика Kubernetes, чтобы эффективно размещать Pods на узлах, а также для kubelet для обеспечения контроля ресурсов. • **Requests**: минимально гарантированный CPU/память, необходимый для **запуска** и выполнения контейнера. Это значение используется

Параметр	Описание
	планировщиком Kubernetes для выбора **Node**, на котором может работать Pod. • **Limits**: максимальное количество CPU/памяти, которое контейнер может использовать во время выполнения. Превышение CPU лимитов приводит к ограничению (throttling), а превышение лимитов памяти — к завершению контейнера (Out Of Memory - OOM Killed). Для подробного описания единиц измерения (например, `m` для milliCPU, `Mi` для мебибайт) смотрите Resource Units. **Логика расчёта ресурсов на уровне Pod** Эффективные значения Requests и Limits для CPU и памяти на уровне Pod вычисляются как сумма и максимум соответствующих значений отдельных контейнеров. Метод расчёта для Requests и Limits аналогичен; в этом документе приведена логика на примере Limits. Если Pod содержит только стандартные контейнеры (бизнес-контейнеры): Эффективное значение CPU/Memory Limit Pod — это сумма CPU/Memory Limit всех контейнеров внутри Pod. **Пример**: Если Pod содержит два контейнера с CPU/Memory Limits 100m/100Mi и 50m/200Mi соответственно, то агрегированное значение CPU/Memory Limit Pod будет 150m/300Mi. Если Pod содержит и initContainers, и стандартные контейнеры: Шаги расчёта CPU/Memory Limit Pod следующие: • 1. Определить максимальное значение CPU/Memory Limit среди всех initContainers. • 2. Вычислить сумму CPU/Memory Limit всех стандартных контейнеров. • 3. Сравнить результаты из шагов 1 и 2. Итоговое значение CPU/Memory Limit Pod — это максимум из CPU значений (максимум initContainers и сумма контейнеров) и максимум из Memory значений (максимум initContainers и сумма контейнеров). **Пример расчёта**: Если Pod содержит два initContainers с CPU/Memory Limits 100m/200Mi и 200m/100Mi, максимальное эффективное значение CPU/Memory Limit для initContainers будет 200m/200Mi. Одновременно, если Pod также содержит два стандартных контейнера с CPU/Memory Limits 100m/100Mi и 50m/200Mi, общая сумма Limit для стандартных контейнеров составит 150m/300Mi. Следовательно, итоговое значение

Параметр	Описание
	CPU/Memory Limit Pod будет max(200m, 150m) для CPU и max(200Mi, 300Mi) для памяти, то есть 200m/300Mi.
Source	Контроллер рабочей нагрузки Kubernetes, управляющий жизненным циклом этого Pod. Это могут быть **Deployments**, **StatefulSets**, **DaemonSets**, **Jobs**.
Restart	Количество перезапусков контейнера внутри **Pod** с момента запуска **Pod**. Высокое количество перезапусков часто указывает на проблему с приложением или его окружением.
Node	Имя Kubernetes Node, на котором в данный момент запущен Pod.
Service Account	Service Account — это объект Kubernetes, который предоставляет идентичность процессам и сервисам, работающим внутри Pod, позволяя им аутентифицироваться и получать доступ к Kubernetes APIServer. Это поле обычно видно только пользователям с ролью администратора платформы или аудитора платформы, что позволяет просматривать YAML-определение Service Account.

Удаление Pod

Удаление Pod может повлиять на работу вычислительных компонентов; пожалуйста, действуйте осторожно.

Сценарии использования

- Быстро восстановить Pod в желаемое состояние: Если Pod остаётся в состоянии, которое влияет на бизнес-процессы, например `Pending` или `CrashLoopBackOff`, ручное удаление Pod после устранения причины ошибки поможет ему быстро вернуться в желаемое состояние, например `Running`. При этом удалённый Pod будет пересоздан на текущем узле или переназначен.
- Очистка ресурсов для управления операциями: Некоторые Pod достигают определённого этапа, когда они больше не изменяются, и такие группы часто накапливаются в большом количестве, усложняя управление другими Pod. К Pod, подлежащим очистке, могут относиться те, что находятся в статусе `Evicted` из-за

нехватки ресурсов узла, или в статусе `Completed`, вызванном повторяющимися запланированными задачами. В этом случае удалённые Pod больше не будут существовать.

Примечание: Для запланированных задач, если необходимо проверять логи каждого выполнения задачи, не рекомендуется удалять соответствующие Pod в статусе `Completed`.

Процедура

1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Workloads** > **Pods**.
3. (Удаление по одному) Нажмите ⋮ справа от Pod, который нужно удалить > **Delete**, подтвердите.
4. (Массовое удаление) Выберите Pod для удаления, нажмите **Delete** над списком, подтвердите.

Контейнеры

Содержание

Понимание контейнеров

Обратитесь к официальной документации сайта Kubernetes: Containers ↗.

Контейнер — это легковесный исполняемый пакет программного обеспечения, который включает всё необходимое для запуска приложения: код, среду выполнения, системные инструменты, системные библиотеки и настройки. Хотя Pods являются наименьшими развертываемыми единицами, контейнеры — это основные компоненты внутри Pods.

Понимание эфемерных контейнеров

Отладка контейнеров с помощью
Обратитесь к официальной документации сайта Kubernetes: Ephemeral Containers ↗

Функция Kubernetes Ephemeral Containers предоставляет надёжный способ отладки запущенных контейнеров путём внедрения специализированных инструментов отладки (системных, сетевых и дисковых утилит) в существующий Pod.

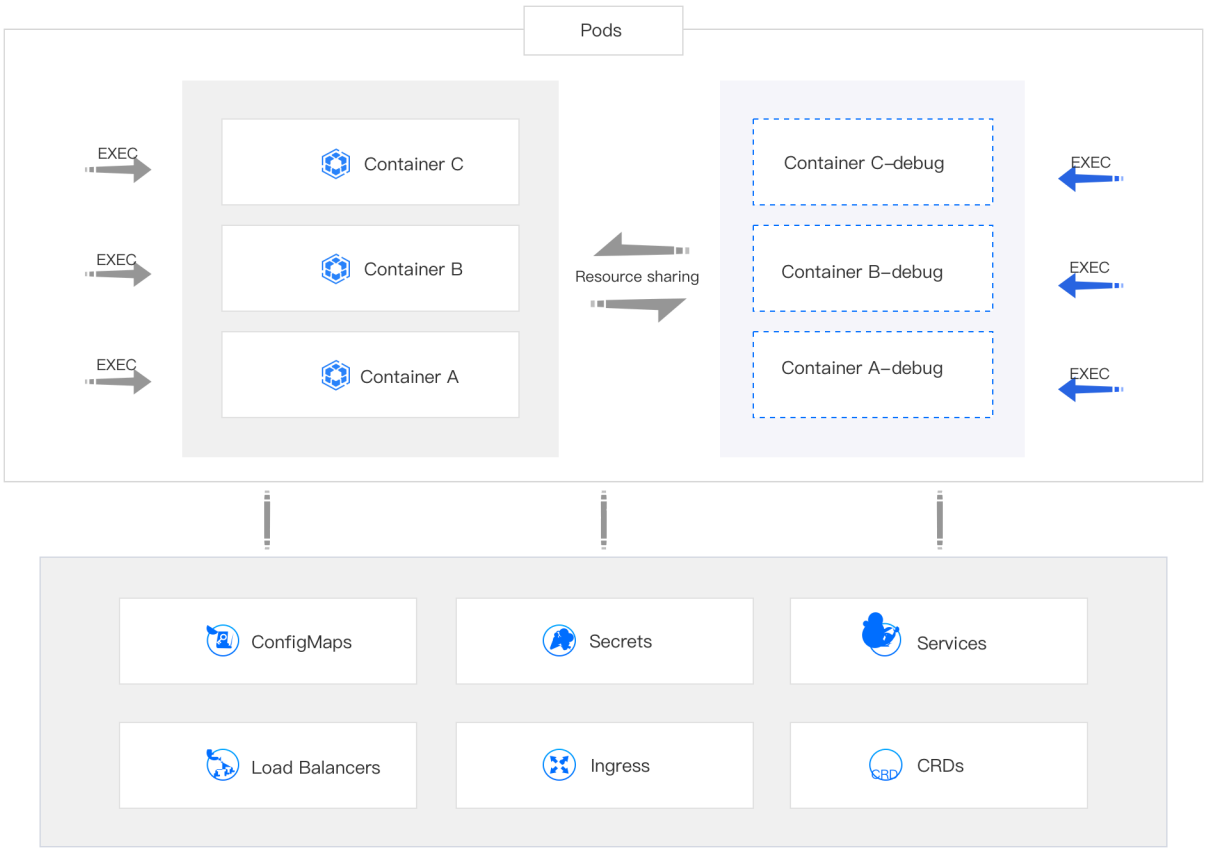
Хотя часто можно выполнять команды непосредственно внутри запущенного контейнера с помощью kubectl exec, многие производственные образы контейнеров намеренно минимальны и могут не содержать необходимых утилит для отладки (например, bash, net-tools, tcpdump), чтобы уменьшить размер образа и поверхность атаки. Эфемерные контейнеры решают эту проблему, предоставляя преднастроенную среду с богатым набором инструментов отладки, что делает их идеальными для следующих сценариев:

- **Диагностика неисправностей**: когда основной контейнер приложения испытывает проблемы (например, неожиданные сбои, ухудшение производительности, проблемы с сетевым подключением), помимо проверки стандартных событий Pod и логов, часто требуется более глубокая интерактивная отладка непосредственно в среде выполнения Pod.
- **Настройка конфигурации и эксперименты**: если текущая конфигурация приложения проявляет неэффективное поведение, вы можете временно изменить настройки компонентов или протестировать новые конфигурации непосредственно в запущенном контейнере, чтобы наблюдать немедленные эффекты и разработать улучшенные решения.

Принцип реализации: использование эфемерных контейнеров

Функциональность отладки реализована с помощью **Ephemeral Containers**. Эфемерный контейнер — это специальный тип контейнера, предназначенный для интроспекции и отладки. Он разделяет сетевое пространство имён Pod и пространство имён процессов (если включено) с существующими основными `containers`, что позволяет ему напрямую взаимодействовать и наблюдать процессы приложения.

Вы можете динамически добавить эфемерный контейнер (например, `my-app-debug`) в запущенный Pod и использовать его предустановленные инструменты отладки. Результаты диагностики из этого эфемерного контейнера напрямую относятся к поведению и состоянию основных приложений `containers` внутри того же Pod.



:::Notes * Вы не можете добавить эфемерный контейнер, напрямую изменяя статический манифест Pod (PodSpec). Функция Ephemeral Containers предназначена для динамического внедрения в запущенные Pods, обычно через API-вызовы (например, kubectl debug). * **Ephemeral Containers**, созданные через функцию отладки, не имеют гарантий ресурсов (CPU/память) или планирования (то есть они не блокируют запуск Pod и не получают собственный класс QoS) и не будут автоматически перезапускаться при выходе. Поэтому избегайте запуска в них постоянных бизнес-приложений; они предназначены исключительно для отладки. * Будьте осторожны при использовании функции отладки, если узел (Node), на котором расположен Pod, испытывает высокую загрузку ресурсов или близок к исчерпанию ресурсов. Внедрение эфемерного контейнера, даже с минимальным потреблением ресурсов, может способствовать эвакуации Pod при сильном давлении на ресурсы. :::

Отладка эфемерных контейнеров с помощью CLI

В Kubernetes 1.25+ доступна команда `kubectl debug` для создания эфемерных контейнеров. Этот метод предоставляет мощную альтернативу командной строки для

отладки.

Команда

```
kubectl debug -it <pod-name> --image=<debug-image> --target=<target-container-name> -n <namespace>
# --image: Указывает образ для отладки (например, busybox, ubuntu, nicolaka/netshoot) с необходимыми инструментами.
# --target: (Опционально) Указывает имя контейнера в Pod, на который направлена отладка. Если опущено и контейнер один — используется он. Если несколько — по умолчанию первый.
# -n: Указывает namespace.
```

Пример YAML файла Pod

Пример: Отладка `nginx` в `my-nginx-pod`

- Сначала убедитесь, что Pod запущен:

  ```
  kubectl apply -f pod-example.yaml
  ```

- Теперь создайте эфемерный контейнер отладки с именем `debugger` внутри `my-nginx-pod`, нацеленный на my-nginx-container, используя образ `busybox`:

  ```
  kubectl debug -it my-nginx-pod --image=busybox --target=nginx -- /bin/sh
  ```

  Эта команда подключит вас к shell внутри эфемерного контейнера debugger. Теперь вы можете использовать инструменты busybox для отладки my-nginx-container.

- Чтобы просмотреть эфемерные контейнеры, прикреплённые к Pod:

  ```
  kubectl describe pod my-nginx-pod
  ```

  Обратите внимание на раздел `Ephemeral Containers` в выводе.

Отладка эфемерных контейнеров через веб-консоль

1. Перейдите в **Container Platform**, затем в левом меню выберите **Workloads** > **Pods**.
2. Найдите нужный Pod и нажмите ⋮ > **Debug**.
3. Выберите конкретный контейнер внутри Pod, который хотите отлаживать.
4. (Опционально) Если интерфейс предложит **необходимость инициализации** (например, для настройки среды отладки), нажмите **Initialize**.

> **INFO**
>
> После инициализации функции Debug, пока Pod не будет пересоздан, вы можете напрямую войти в эфемерный контейнер (например, *Container A-debug*) для отладки.

5. Дождитесь готовности окна терминала отладки и начните операции отладки.
   Совет: Нажмите на опцию "Command Query" в правом верхнем углу терминала, чтобы увидеть список распространённых инструментов отладки и примеры их использования.

> **INFO**
>
> Нажмите на запрос команд в правом верхнем углу, чтобы просмотреть распространённые инструменты и их использование.

6. По окончании отладки закройте окно терминала.

Взаимодействие с контейнерами

Вы можете напрямую взаимодействовать с внутренним экземпляром запущенного контейнера с помощью команды `kubectl exec`, что позволяет выполнять произвольные операции в командной строке. Кроме того, Kubernetes предоставляет удобные функции для загрузки и скачивания файлов в контейнер и из него.

Взаимодействие с контейнерами через CLI

Exec

Для выполнения команды внутри конкретного контейнера в Pod (полезно для получения shell, запуска диагностических команд и т. п.):

```
kubectl exec -it <pod-name> -c <container-name> -n <namespace> -- <comman
d>
# -it: Обеспечивает интерактивный режим и TTY (псевдотерминал) для сессии
shell.
# -c: Указывает имя целевого контейнера в Pod. Опускается, если в Pod оди
н контейнер.
# --: Разделяет аргументы kubectl и команду для выполнения в контейнере.
```

- **Пример**: Получение Bash shell в `nginx` из `my-nginx-pod`

  ```
  kubectl exec -it my-nginx-pod -c nginx -n default -- /bin/bash
  ```

- **Пример**: Просмотр файлов в `/tmp` контейнера

  ```
  kubectl exec my-nginx-pod -c nginx -n default -- ls /tmp
  ```

Передача файлов

- Чтобы скопировать файлы с локальной машины в контейнер внутри Pod:

  ```
  kubectl cp <local-file-path> <namespace>/<pod-name>:<container-file-pat
  h> -c <container-name>
  # -c: (Опционально) Указывает целевой контейнер, если в Pod несколько к
  онтейнеров.

  # Пример: Загрузка `my-config.txt` в директорию HTML Nginx
  kubectl cp my-config.txt default/my-nginx-pod:/usr/share/nginx/html/my-
  config.txt -c nginx
  ```

- Чтобы скопировать файлы из контейнера внутри Pod на локальную машину:

```
kubectl cp <namespace>/<pod-name>:<container-file-path> <local-file-path> -c <container-name>

# Пример: Загрузка логов доступа Nginx
kubectl cp default/my-nginx-pod:/var/log/nginx/access.log ./nginx_access.log -c nginx
```

Взаимодействие с контейнерами через веб-консоль

Вход в контейнер через Applications

Вы можете войти во внутренний экземпляр контейнера с помощью команды `kubectl exec`, что позволяет выполнять операции в командной строке в окне веб-консоли. Кроме того, можно легко загружать и скачивать файлы внутри контейнера с помощью функции передачи файлов.

1. Перейдите в **Container Platform**, затем в левом меню выберите **Application** > **Applications**.
2. Нажмите на ***Application Name***.
3. Найдите связанный workload (например, Deployment, StatefulSet), нажмите **EXEC**, затем выберите конкретный *Pod Name*, в который хотите войти. **EXEC** > ***Container Name***.
4. Введите команду, которую хотите выполнить.
5. Нажмите **OK**, чтобы войти в окно веб-консоли и выполнять операции в командной строке.
6. Нажмите **File Transfer**.
   - Введите **Upload Path** для загрузки локальных файлов в контейнер (например, конфигурационные файлы для тестирования).
   - Введите **Download Path** для скачивания логов, диагностических данных или других файлов из контейнера на локальную машину для анализа.

Вход в контейнер через Pod

1. Перейдите в **Container Platform**, затем в левом меню выберите **Workloads** > **Pods**.

2. Найдите нужный Pod, нажмите вертикальное многоточие (⋮) рядом с ним, выберите EXEC, затем выберите конкретный Container Name внутри этого Pod, в который хотите войти.
3. Введите команду, которую хотите выполнить.
4. Нажмите **OK**, чтобы войти в окно веб-консоли и выполнять операции в командной строке.
5. Нажмите **File Transfer**.

   - Введите **Upload Path** для загрузки локальных файлов в контейнер (например, конфигурационные файлы для тестирования).
   - Введите **Download Path** для скачивания логов, диагностических данных или других файлов из контейнера на локальную машину для анализа.

Работа с Helm charts

Содержание

---

1. Понимание Helm

Helm — это менеджер пакетов, который упрощает развертывание приложений и сервисов в кластерах Alauda Container Platform.
Helm использует формат упаковки, называемый *charts*. Helm chart — это набор файлов, описывающих ресурсы Kubernetes.
Создание chart в кластере порождает экземпляр chart, называемый *release*.
Каждый раз при создании chart, обновлении или откате release создаётся инкрементальная ревизия.

1.1. Основные возможности

Helm предоставляет возможность:

- Искать большое количество charts в репозиториях charts
- Модифицировать существующие charts
- Создавать собственные charts с использованием ресурсов Kubernetes
- Упаковывать приложения и делиться ими в виде charts

1.2. Каталог

Каталог построен на базе Helm и представляет собой комплексную платформу управления распространением Chart, расширяющую ограничения инструмента Helm CLI. Платформа позволяет разработчикам удобнее управлять, развертывать и использовать charts через удобный интерфейс.

Определения терминов

Термин	Определение	Примечания
Application Catalog	Универсальная платформа управления Helm Charts	
Helm Charts	Формат упаковки приложений	
HelmRequest	CRD. Определяет конфигурацию, необходимую для развертывания Helm Chart	Template Application
ChartRepo	CRD. Соответствует репозиторию Helm charts	Template Repository
Chart	CRD. Соответствует Helm Charts	Template

1.3 Понимание HelmRequest

В Alauda Container Platform развертывания Helm в основном управляются через кастомный ресурс **HelmRequest**. Такой подход расширяет стандартный функционал Helm и интегрирует его в нативную модель ресурсов Kubernetes.

Отличия HelmRequest от Helm

Стандартный Helm использует CLI-команды для управления release, тогда как Alauda Container Platform применяет ресурсы HelmRequest для определения, развертывания и управления Helm charts. Основные отличия:

1. **Декларативный vs Императивный**: HelmRequest обеспечивает декларативный подход к развертываниям Helm, в то время как традиционный Helm CLI — императивный.
2. **Нативность Kubernetes**: HelmRequest — это кастомный ресурс, напрямую интегрированный с Kubernetes API.
3. **Непрерывная синхронизация**: Captain постоянно отслеживает и синхронизирует ресурсы HelmRequest с их желаемым состоянием.

4. **Поддержка мультикластерности**: HelmRequest поддерживает развертывания в нескольких кластерах через платформу.
5. **Интеграция с функциями платформы**: HelmRequest может интегрироваться с другими функциями платформы, например, с ресурсами Application.

Интеграция HelmRequest и Application

Ресурсы HelmRequest и Application концептуально схожи, и пользователи могут захотеть видеть их единообразно. Платформа предоставляет механизм синхронизации HelmRequest как ресурсов Application.

Пользователи могут пометить HelmRequest для развертывания как Application, добавив следующую аннотацию:

```
alauda.io/create-app: "true"
```

При включении этой функции в UI платформы отображаются дополнительные поля и ссылки на соответствующую страницу Application.

Рабочий процесс развертывания

Рабочий процесс развертывания charts через HelmRequest включает:

1. **Пользователь** создаёт или обновляет ресурс HelmRequest
2. **HelmRequest** содержит ссылки на chart и значения для применения
3. **Captain** обрабатывает HelmRequest и создаёт Helm Release
4. **Release** содержит развернутые ресурсы
5. **Metis** отслеживает HelmRequest с аннотациями приложения и синхронизирует их с Applications
6. **Application** предоставляет единый обзор развернутых ресурсов

Определения компонентов

- **HelmRequest**: Определение кастомного ресурса, описывающего желаемое развертывание Helm chart

- **Captain**: Контроллер, обрабатывающий ресурсы HelmRequest и управляющий Helm release (исходный код доступен по адресу https://github.com/alauda/captain↗)
- **Release**: Развернутый экземпляр Helm chart
- **Charon**: Компонент, отслеживающий HelmRequest и создающий соответствующие ресурсы Application
- **Application**: Унифицированное представление развернутых ресурсов с дополнительными возможностями управления
- **Archon-api**: Компонент, отвечающий за специфические расширенные функции API внутри платформы

2 Развертывание Helm Charts как приложений через CLI

2.1 Обзор рабочего процесса

Подготовка chart → Упаковка chart → Получение API токена → Создание репозитория chart → Загрузка chart → Загрузка связанных образов → Развертывание приложения → Обновление приложения → Удаление приложения → Удаление репозитория chart

2.2 Подготовка Chart

Helm использует формат упаковки, называемый charts. Chart — это набор файлов, описывающих ресурсы Kubernetes. Один chart может использоваться для развертывания всего — от простого pod до сложного стека приложений.

См. официальную документацию: Helm Charts Documentation↗

Пример структуры каталога chart:

```
nginx/
├── Chart.lock
├── Chart.yaml
├── README.md
├── charts/
│   └── common/
│       ├── Chart.yaml
│       ├── README.md
│       ├── templates/
│       │   ├── _affinities.tpl
│       │   ├── _capabilities.tpl
│       │   ├── _errors.tpl
│       │   ├── _images.tpl
│       │   ├── _ingress.tpl
│       │   ├── _labels.tpl
│       │   ├── _names.tpl
│       │   ├── _secrets.tpl
│       │   ├── _storage.tpl
│       │   ├── _tplvalues.tpl
│       │   ├── _utils.tpl
│       │   ├── _warnings.tpl
│       │   └── validations/
│       │       ├── _cassandra.tpl
│       │       ├── _mariadb.tpl
│       │       ├── _mongodb.tpl
│       │       ├── _postgresql.tpl
│       │       ├── _redis.tpl
│       │       └── _validations.tpl
│       └── values.yaml
├── ci/
│   ├── ct-values.yaml
│   └── values-with-ingress-metrics-and-serverblock.yaml
├── templates/
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── extra-list.yaml
│   ├── health-ingress.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── ldap-daemon-secrets.yaml
│   ├── pdb.yaml
│   ├── server-block-configmap.yaml
```

```
|   ├── serviceaccount.yaml
|   ├── servicemonitor.yaml
|   ├── svc.yaml
|   └── tls-secrets.yaml
├── values.descriptor.yaml
├── values.schema.json
└── values.yaml
```

Описание ключевых файлов:

- `values.descriptor.yaml` (опционально): Работает с ACP UI для отображения удобных форм
- `values.schema.json` (опционально): Валидирует содержимое values.yaml и отображает простой UI
- `values.yaml` (обязательно): Определяет параметры развертывания chart

2.3 Упаковка Chart

Используйте команду `helm package` для упаковки chart:

```
helm package nginx
# 输出: Successfully packaged chart and saved it to: /charts/nginx-8.8.0.tgz
```

2.4 Получение API токена

1. В **Alauda Container Platform** нажмите на аватар в правом верхнем углу → **Profile**
2. Нажмите **Add Api Token**
3. Введите подходящее описание и срок действия
4. Сохраните отображённый токен (показывается только один раз)

2.5 Создание репозитория Chart

Создайте локальный репозиторий chart через API:

```
curl -k --request POST \
--url https://$ACP_DOMAIN/catalog/v1/chartrepos \
--header 'Authorization:Bearer $API_TOKEN' \
--header 'Content-Type: application/json' \
--data '{
  "apiVersion": "v1",
  "kind": "ChartRepoCreate",
  "metadata": {
    "name": "test",
    "namespace": "cpaas-system"
  },
  "spec": {
    "chartRepo": {
      "apiVersion": "app.alauda.io/v1beta1",
      "kind": "ChartRepo",
      "metadata": {
        "name": "test",
        "namespace": "cpaas-system",
        "labels": {
          "project.cpaas.io/catalog": "true"
        }
      },
      "spec": {
        "type": "Local",
        "url": null,
        "source": null
      }
    }
  }
}'
```

2.6 Загрузка Chart

Загрузите упакованный chart в репозиторий:

```
curl -k --request POST \
--url https://$ACP_DOMAIN/catalog/v1/chartrepos/cpaas-system/test/charts
\
--header 'Authorization:Bearer $API_TOKEN' \
--data-binary @"/root/charts/nginx-8.8.0.tgz"
```

2.7 Загрузка связанных образов

1. Скачайте образ: `podman pull nginx`
2. Сохраните в tar-пакет: `podman save nginx > nginx.latest.tar`
3. Загрузите и отправьте в приватный реестр:

```
podman load -i nginx.latest.tar
podman tag nginx:latest 192.168.80.8:30050/nginx:latest
podman push 192.168.80.8:30050/nginx:latest
```

2.8 Развертывание приложения

Создайте ресурс Application через API:

```
curl -k --request POST \
--url https://$ACP_DOMAIN/acp/v1/kubernetes/$CLUSTER_NAME/namespaces/$NAM
ESPACE/applications \
--header 'Authorization:Bearer $API_TOKEN' \
--header 'Content-Type: application/json' \
--data '{
  "apiVersion": "app.k8s.io/v1beta1",
  "kind": "Application",
  "metadata": {
    "name": "test",
    "namespace": "catalog-ns",
    "annotations": {
      "app.cpaas.io/chart.source": "test/nginx",
      "app.cpaas.io/chart.version": "8.8.0",
      "app.cpaas.io/chart.values": "{\"image\":{\"pullPolicy\":\"IfNotPre
sent\"}}"
    },
    "labels": {
      "sync-from-helmrequest": "true"
    }
  }
}'
```

2.9 Обновление приложения

Обновите приложение с помощью PATCH-запроса:

```
curl -k --request PATCH \
--url https://$ACP_DOMAIN/acp/v1/kubernetes/$CLUSTER_NAME/namespaces/$NAM
ESPACE/applications/test \
--header 'Authorization:Bearer $API_TOKEN' \
--header 'Content-Type: application/merge-patch+json' \
--data '{
  "apiVersion": "app.k8s.io/v1beta1",
  "kind": "Application",
  "metadata": {
    "annotations": {
      "app.cpaas.io/chart.values": "{\"image\":{\"pullPolicy\":\"Always
\"}}"
    }
  }
}'
```

2.10 Удаление приложения

Удалите ресурс Application:

```
curl -k --request DELETE \
--url https://$ACP_DOMAIN/acp/v1/kubernetes/$CLUSTER_NAME/namespaces/$NAM
ESPACE/applications/test \
--header 'Authorization:Bearer $API_TOKEN'
```

2.11 Удаление репозитория Chart

```
curl -k --request DELETE \
--url https://$ACP_DOMAIN/apis/app.alauda.io/v1beta1/namespaces/cpaas-sys
tem/chartrepos/test \
--header 'Authorization:Bearer $API_TOKEN'
```

3. Развертывание Helm Charts как приложений через UI

3.1 Обзор рабочего процесса

Добавление шаблонов в управляемые репозитории → Загрузка шаблонов → Управление версиями шаблонов

3.2 Предварительные требования

Репозитории шаблонов добавляются администраторами платформы. Пожалуйста, обратитесь к администратору платформы для получения названий доступных репозиториев шаблонов типа Chart или OCI Chart с правами **Management**.

3.3 Добавление шаблонов в управляемые репозитории

1. Перейдите в **Catalog**.
2. В левой навигационной панели нажмите **Helm Charts**.
3. Нажмите **Add Template** в правом верхнем углу страницы и выберите репозиторий шаблонов согласно параметрам ниже.

Параметр	Описание
Template Repository	Синхронизирует шаблон напрямую с репозиторием шаблонов типа Chart или OCI Chart с правами **Management**. Владельцы проектов, назначенные на этот **Template Repository**, могут напрямую использовать шаблон.
Template Directory	При выборе типа репозитория шаблонов OCI Chart необходимо выбрать или вручную ввести директорию для хранения Helm Chart. **Примечание**: При ручном вводе новой директории платформа создаст эту директорию в репозитории шаблонов, но существует риск неудачи создания.

4. Нажмите **Upload Template** и загрузите локальный шаблон в репозиторий.
5. Нажмите **Confirm**. Процесс загрузки шаблона может занять несколько минут, пожалуйста, подождите.
   Примечание: Когда статус шаблона изменится с `Uploading` на `Upload Successful`, это означает успешную загрузку шаблона.
6. Если загрузка не удалась, устраните неполадки согласно появившимся подсказкам.
   Примечание: Неправильный формат файла означает, что в загруженном архиве есть проблемы с файлами, например, отсутствует содержимое или неверное форматирование.

3.4 Удаление конкретных версий шаблонов

Если версия шаблона больше не актуальна, её можно удалить.

Шаги для выполнения

1. Перейдите в **Catalog**.
2. В левой навигационной панели нажмите **Helm Charts**.
3. Нажмите на карточку Chart для просмотра деталей.
4. Нажмите **Manage Versions**.
5. Найдите устаревший шаблон, нажмите **Delete** и подтвердите.
   После удаления версии соответствующее приложение не сможет быть обновлено.

Конфигурации

Настройка ConfigMap

- Понимание ConfigMap
- Ограничения ConfigMap
- Пример ConfigMap
- Создание ConfigMap через веб-консоль
- Создание ConfigMap с помощью CLI
- Операции
- Просмотр, редактирование и удаление
- Способы использования ConfigMap в Pc
- ConfigMap и Secret

Настройка Secrets

- Понимание Secrets
- Создание Secret типа Opaque
- Создание Secret для реестра контейнеров
- Создание Secret типа Basic Auth
- Создание Secret типа SSH-Auth
- Создание Secret типа TLS
- Создание Secret через веб-консоль
- Как использовать Secret в Pod
- Последующие действия
- Операции

Настройка ConfigMap

ConfigMap позволяет отделить артефакты конфигурации от содержимого образа, чтобы обеспечить переносимость контейнеризованных приложений. В следующих разделах описывается, что такое ConfigMap и как его создавать и использовать.

Содержание

Понимание ConfigMap

Многие приложения требуют настройки с помощью комбинации конфигурационных файлов, аргументов командной строки и переменных окружения. В OpenShift Container Platform эти артефакты конфигурации отделены от содержимого образа, чтобы обеспечить переносимость контейнеризованных приложений.

Объект `ConfigMap` предоставляет механизмы для внедрения данных конфигурации в контейнеры, при этом контейнеры остаются независимыми от OpenShift Container Platform. ConfigMap может использоваться для хранения как мелкозернистой информации, например отдельных свойств, так и крупнозернистой, например целых конфигурационных файлов или JSON-блоков.

Объект `ConfigMap` содержит пары ключ-значение с данными конфигурации, которые могут использоваться в подах или хранить данные конфигурации для системных компонентов, таких как контроллеры. Например:

```
# my-app-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
data:
  app_mode: "development"
  feature_flags: "true"
  database.properties: |-
    jdbc.url=jdbc:mysql://localhost:3306/mydb
    jdbc.username=user
    jdbc.password=password
  log_settings.json: |-
    {
      "level": "INFO",
      "format": "json"
    }
```

Примечание: Вы можете использовать поле `binaryData` при создании ConfigMap из бинарного файла, например, изображения.

Данные конфигурации могут использоваться в подах различными способами. ConfigMap может быть использован для:

- Заполнения значений переменных окружения в контейнерах

- Установки аргументов командной строки в контейнере
- Заполнения конфигурационных файлов в томе

Пользователи и системные компоненты могут хранить данные конфигурации в ConfigMap. ConfigMap похож на secret, но предназначен для более удобной работы со строками, не содержащими конфиденциальной информации.

Ограничения ConfigMap

- ConfigMap должен быть создан до того, как его содержимое может быть использовано в подах.
- Контроллеры могут быть написаны с учетом отсутствия данных конфигурации. Обращайтесь к документации отдельных компонентов, настроенных с помощью ConfigMap, в каждом конкретном случае.
- Объекты `ConfigMap` находятся в проекте.
- Они могут ссылаться только на поды в том же проекте.
- Kubectl поддерживает использование ConfigMap только для подов, полученных от API-сервера. Это включает поды, созданные с помощью CLI, или косвенно через replication controller. Это не включает поды, созданные с помощью флага `--manifest-url` узла OpenShift Container Platform, его флага `--config` или REST API, поскольку это не стандартные способы создания подов.

> **NOTE**
>
> Под может использовать ConfigMap только в пределах одного namespace.

Пример ConfigMap

Теперь вы можете использовать app-config в `Pod`.

```yaml
# app-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_ENV: "production"
  LOG_LEVEL: "debug"
```

Создание ConfigMap через веб-консоль

1. Перейдите в **Container Platform**.
2. В левой боковой панели выберите **Configuration** > **ConfigMap**.
3. Нажмите **Create ConfigMap**.
4. Следуйте инструкциям ниже для настройки соответствующих параметров.

Параметр	Описание
Entries	Относится к парам `ключ:значение`, поддерживает методы **Добавить** и **Импортировать**. • **Добавить**: Вы можете добавлять элементы конфигурации по одному или вставить одну или несколько строк с парами key=value в поле ввода ключа для массового добавления. • **Импортировать**: Импортируйте текстовый файл размером не более 1M. Имя файла будет использоваться как ключ, а содержимое файла — как значение, заполненное в элемент конфигурации.
Binary Entries	Относится к бинарным файлам размером не более 1M. Имя файла будет использоваться как ключ, а содержимое файла — как значение, заполненное в элемент конфигурации. **Примечание**: После создания ConfigMap импортированные файлы нельзя изменять.

Пример формата массового добавления:

```
# Одна пара key=value на строку, несколько пар должны быть на отдельных строках, иначе они не будут корректно распознаны после вставки.
key1=value1
key2=value2
key3=value3
```

5. Нажмите **Create**.

Создание ConfigMap с помощью CLI

```
kubectl create configmap app-config \
  --from-literal=APP_ENV=production \
  --from-literal=LOG_LEVEL=debug
```

Или из файла:

```
kubectl apply -f app-config.yaml
```

Операции

Вы можете нажать (⋮) справа на странице списка или выбрать **Actions** в правом верхнем углу страницы с деталями, чтобы при необходимости обновить или удалить ConfigMap.

Изменения в ConfigMap повлияют на рабочие нагрузки, которые ссылаются на эту конфигурацию, поэтому заранее ознакомьтесь с инструкциями по эксплуатации.

Операция	Описание
Обновить	• После добавления или обновления ConfigMap все рабочие нагрузки, которые ссылались на этот ConfigMap (или его элементы конфигурации) через переменные окружения, должны пересоздать свои поды, чтобы изменения вступили в силу.

Операция	Описание
	• Для импортированных бинарных элементов конфигурации поддерживается только обновление ключей, а не значений.
Удалить	После удаления ConfigMap рабочие нагрузки, которые ссылались на этот ConfigMap (или его элементы конфигурации) через переменные окружения, могут столкнуться с проблемами при создании подов, если они будут пересозданы и не смогут найти источник ссылки.

Просмотр, редактирование и удаление через CLI

```
kubectl get configmap app-config -o yaml
kubectl edit configmap app-config
kubectl delete configmap app-config
```

Способы использования ConfigMap в Pod

В виде переменных окружения

```
envFrom:
  - configMapRef:
      name: app-config
```

Каждый ключ становится переменной окружения в контейнере.

В виде файлов в томе

```yaml
volumes:
  - name: config-volume
    configMap:
      name: app-config

volumeMounts:
  - name: config-volume
    mountPath: /etc/config
```

Каждый ключ — это файл в каталоге `/etc/config`, а содержимое файла — значение.

В виде отдельных переменных окружения

```yaml
env:
  - name: APP_ENV
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: APP_ENV
```

ConfigMap и Secret

Особенность	ConfigMap	Secret
Тип данных	Не конфиденциальные	Конфиденциальные (например, пароли)
Кодирование	Обычный текст	Base64-кодирование
Сценарии использования	Конфигурации, флаги	Пароли, токены

Настройка Secrets

Содержание

Понимание Secrets

В Kubernetes (k8s) Secret — это базовый объект, предназначенный для хранения и управления конфиденциальной информацией, такой как пароли, OAuth-токены, SSH-

ключи, TLS-сертификаты и API-ключи. Его основная задача — предотвратить прямое включение чувствительных данных в определения Pod или образы контейнеров, что повышает безопасность и портативность.

Secrets похожи на ConfigMaps, но предназначены специально для конфиденциальных данных. Обычно они хранятся в виде base64-кодированных значений и могут использоваться подами различными способами, включая монтирование в виде томов или предоставление в виде переменных окружения.

Характеристики использования

- **Повышенная безопасность**: По сравнению с конфигурационными картами в открытом виде (Kubernetes ConfigMap), Secrets обеспечивают лучшую защиту, храня чувствительные данные в Base64-кодировке. Этот механизм в сочетании с возможностями Kubernetes по контролю доступа значительно снижает риск утечки данных.
- **Гибкость и управление**: Использование Secrets предоставляет более безопасный и гибкий подход, чем жесткое кодирование конфиденциальной информации непосредственно в файлах определения Pod или образах контейнеров. Такое разделение упрощает управление и изменение чувствительных данных без необходимости менять код приложения или образы контейнеров.

Поддерживаемые типы

Kubernetes поддерживает различные типы Secrets, каждый из которых предназначен для конкретных сценариев использования. Обычно платформа поддерживает следующие типы:

- **Opaque**: универсальный тип Secret для хранения произвольных пар ключ-значение с конфиденциальными данными, такими как пароли или API-ключи.
- **TLS**: специально предназначен для хранения сертификатов и приватных ключей TLS (Transport Layer Security), часто используемых для HTTPS и безопасного ingress.
- **SSH Key**: используется для хранения приватных SSH-ключей, например, для безопасного доступа к Git-репозиториям или другим сервисам с поддержкой SSH.
- **SSH Authentication (kubernetes.io/ssh-auth)**: хранит данные аутентификации для передачи данных по протоколу SSH.

- **Username/Password (kubernetes.io/basic-auth)**: используется для хранения учетных данных базовой аутентификации (имя пользователя и пароль).
- **Image Pull Secret**: хранит JSON-строку аутентификации, необходимую для загрузки контейнерных образов из приватных репозиториев.

Способы использования

Secrets могут использоваться приложениями внутри подов различными способами:

- **В виде переменных окружения**: конфиденциальные данные из Secret могут быть внедрены непосредственно в переменные окружения контейнера.
- **В виде монтируемых файлов (томов)**: Secrets могут монтироваться как файлы в том пода, позволяя приложениям читать конфиденциальные данные по заданному пути.

Примечание: Экземпляры Pod в рамках workload могут ссылаться только на Secrets в том же namespace. Для расширенного использования и YAML-конфигураций обратитесь к официальной документации Kubernetes↗.

Создание Secret типа Opaque

```
kubectl create secret generic my-secret \
  --from-literal=username=admin \
  --from-literal=password=Pa$$w0rd
```

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: YWRtaW4= # base64 от "admin"
  password: UGEkJHcwcmQ= # base64 от "Pa$$w0rd"
```

Вы можете декодировать их так:

```
echo YWRtaW4= | base64 --decode  # вывод: admin
```

Создание Secret для реестра контейнеров

```
kubectl create secret docker-registry my-container-registry-creds \
  --docker-username=myuser \
  --docker-password=mypass \
  --docker-server=https://registry.example.com \
  --docker-email=my@example.com
```

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: my-container-registry-creds
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8i
OnsidXNlcm5hbWUiOiJteXVzZXIiLCJwYXNzd29yZCI6Im15cGFzcyIsImVtYWlsIjoibXlAZ
XhhbXBsZS5jb20iLCJhdXRoIjoiYlhsMWMyVnlPbTE1Y0dGemN3PT0ifX19
```

K8s автоматически преобразует ваше имя пользователя, пароль, email и информацию о сервере в стандартный формат логина:

```
{
  "auths": {
    "https://registry.example.com": {
      "username": "myuser",
      "password": "mypass",
      "email": "my@example.com",
      "auth": "bXl1c2VyOm15cGFzcw==" # base64(username:password)
    }
  }
}
```

Этот JSON затем кодируется в base64 и используется как значение поля data в Secret.

Используйте его в Pod:

```
imagePullSecrets:
  - name: my-container-registry-creds
```

Создание Secret типа Basic Auth

```
apiVersion: v1
kind: Secret
metadata:
  name: basic-auth-secret
type: kubernetes.io/basic-auth
stringData:
  username: myuser
  password: mypass
```

Создание Secret типа SSH-Auth

Сценарий использования: хранение приватных SSH-ключей (например, для доступа к Git).

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-key-secret
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    ...
    -----END OPENSSH PRIVATE KEY-----
```

Создание Secret типа TLS

Сценарий использования: TLS-сертификаты (используются Ingress, webhook и др.)

```
kubectl create secret tls tls-secret \
--cert=path/to/tls.crt \
--key=path/to/tls.key
```

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: <base64>
  tls.key: <base64>
```

Создание Secret через веб-консоль

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Configuration** > **Secrets**.

3. Нажмите **Create Secret**.
4. Настройте параметры.

   Примечание: В форме ввода конфиденциальные данные, такие как имя пользователя и пароль, автоматически кодируются в формат Base64 перед сохранением в Secret. Преобразованные данные можно просмотреть в YAML-виде.
5. Нажмите **Create**.

Как использовать Secret в Pod

В виде переменных окружения

```
env:
  - name: DB_USERNAME
    valueFrom:
      secretKeyRef:
        name: my-secret
        key: username
```

Из секрета с именем `my-secret` берется значение по ключу `username` и присваивается переменной окружения `DB_USERNAME`.

В виде монтируемых файлов (томов)

```
volumes:
  - name: secret-volume
    secret:
      secretName: my-secret

volumeMounts:
  - name: secret-volume
    mountPath: "/etc/secret"
```

Последующие действия

При создании workloads для нативных приложений в том же namespace вы можете ссылаться на уже созданные Secrets.

Операции

Вы можете нажать (⋮) справа на странице списка или выбрать **Actions** в правом верхнем углу страницы деталей для обновления или удаления Secret по мере необходимости.

Операция	Описание
Обновить	После добавления или обновления Secret, workloads, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, должны пересоздать свои Pod для применения новой конфигурации.
Удалить	• После удаления Secret workloads, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, могут столкнуться с проблемами из-за отсутствия источника ссылки при пересоздании Pod. • Пожалуйста, не удаляйте Secrets, автоматически созданные платформой, так как это может нарушить работу платформы. Например: Secrets типа service-account-token, содержащие данные аутентификации для ресурсов namespace, и Secrets в системных namespace (таких как kube-system).

Наблюдаемость приложений

Мониторинговые панели

- Поддерживается просмотр данных мониторинга ресурсов для компонентов нагрузки на платформе за последние 7 дней (с настраиваемым периодом хранения данных мониторинга). Включает статистику по приложениям, нагрузкам, подам, а также тенденции/рейтинги использования CPU/памяти.
- Поддерживается мониторинг на уровне Namespace.
- Поддерживаемый мониторинг на уровне нагрузки: **Applications**, **Deployments**, **DaemonSets**, **StatefulSets** и **Pods**

Содержание

Предварительные требования

- Установка плагинов мониторинга

Панели мониторинга на уровне Namespace

Процедура

1. В **Container Platform** нажмите **Observe** > **Dashboards**.
2. Просмотрите данные мониторинга в рамках namespace. Предоставляются три панели: **Applications Overview**, **Workloads Overview** и **Pods Overview**.
3. Переключайтесь между панелями для мониторинга целевого **Overview**.

Создание панели мониторинга на уровне Namespace

1. **Администратор** создает выделенную панель мониторинга, руководствуясь инструкцией Создание панели мониторинга.
2. Настройте следующие метки для отображения панели мониторинга на уровне Namespace в **Container Platform**:
   - `cpaas.io/dashboard.folder: container-platform`
   - `cpaas.io/dashboard.tag.overview: "true"`

Мониторинг на уровне нагрузки

> В этой процедуре показано, как просматривать мониторинг подов через интерфейс Deployment.

Панель мониторинга по умолчанию

Процедура

1. В **Container Platform** нажмите **Workloads** > **Deployments**.
2. Выберите имя Deployment из списка.

3. Перейдите на вкладку **Monitoring** для просмотра метрик мониторинга по умолчанию.

Интерпретация метрик

Ресурс мониторинга	Гранулярность метрики	Техническое определение
CPU	**Utilization/Usage**	**Utilization** = Usage/Limit (лимиты) Оценка конфигурации лимитов контейнера. Высокая загрузка указывает на недостаточные лимиты. **Usage** — фактическое потребление ресурсов.
Memory	**Utilization/Usage**	**Utilization** = Usage/Limit (лимиты) Метод оценки аналогичен CPU. Высокий показатель может вызвать нестабильность компонента.
Network Traffic	**Inflow Rate/Outflow Rate**	Сетевой трафик (байт/сек) входящий/исходящий из подов.
Network Packet	**Receiving Rate/Transmit Rate**	Сетевые пакеты (кол-во/сек), принимаемые/отправляемые подами.
Disk Rate	**Read/Write**	Скорость чтения/записи (байт/сек) смонтированных томов на нагрузку.
Disk IOPS	**Read/Write**	Операции ввода/вывода в секунду (IOPS) смонтированных томов на нагрузку.

Пользовательская панель мониторинга

4. Нажмите **Toggle Icon** для переключения на пользовательские панели.
   Руководствуйтесь инструкцией Добавление панели в пользовательскую панель для создания выделенной панели мониторинга на уровне нагрузки.

INFO

Наведите курсор на кривые графика, чтобы просмотреть метрики по каждому поду и выражения PromQL в определённые моменты времени. Если количество подов превышает 15, отображаются только топ-15 записей, отсортированных по убыванию.

Логи

Агрегируйте логи контейнерного рантайма с возможностями визуального запроса. Когда приложения, рабочие нагрузки или другие ресурсы проявляют аномальное поведение, анализ логов помогает диагностировать первопричины.

Содержание

Процедура

> В этой процедуре показано, как просматривать логи контейнерного рантайма через интерфейс Deployment.

1. В **Container Platform** нажмите **Workloads** > **Deployments**.
2. Выберите имя Deployment из списка.
3. Перейдите на вкладку **Logs** для просмотра подробных записей.

Операция	Описание
Pod/Container	Переключайтесь между Pod и Container с помощью выпадающего селектора для просмотра соответствующих логов.

Операция	Описание
Previous Logs	Просмотр логов завершённых контейнеров (доступно, если restartCount контейнера > 0).
Lines	Настройка размера буфера отображаемых логов: 1k/10k/100k строк.
Wrap Line	Переключение переноса строк для длинных записей логов (включено по умолчанию).
Find	Полнотекстовый поиск с подсветкой совпадений и навигацией по нажатию Enter.
Raw	Необработанные потоки логов, напрямую захваченные из интерфейсов контейнерного рантайма (CRI) без форматирования, фильтрации или усечения.
Export	Скачивание необработанных логов.
Full Screen	Нажмите на усечённую строку, чтобы просмотреть полный контент в модальном окне.

WARNING

- **Обработка усечения**: Записи логов, превышающие 2000 символов, будут усечены с добавлением многоточия `...`
  - Усечённые части не могут быть найдены с помощью функции поиска на странице.
  - Нажмите на маркер многоточия `...` в усечённых строках, чтобы просмотреть полный контент в модальном окне.
- **Надёжность копирования**: Избегайте прямого копирования из визуального просмотрщика логов при наличии маркеров усечения (...) или ANSI цветовых кодов. Для получения полных логов всегда используйте функции **Export** или **Raw**.
- **Политика хранения**: Живые логи следуют конфигурации ротации логов Kubernetes. Для исторического анализа используйте Logs ↗ .

События

Информация о событиях, генерируемая изменениями состояния ресурсов Kubernetes и обновлениями операционного статуса, с интегрированным визуальным интерфейсом запросов. Когда приложения, рабочие нагрузки или другие ресурсы сталкиваются с исключениями, анализ событий в реальном времени помогает выявить первопричины.

Содержание

Процедура

> В этой процедуре показано, как просматривать события среды выполнения контейнеров через интерфейс Deployment.

1. В **Container Platform** нажмите **Workloads** > **Deployments**.
2. Выберите имя Deployment из списка.
3. Перейдите на вкладку **Events** для просмотра подробных записей.

Интерпретация записей событий

Записи событий ресурсов: Ниже панели с кратким описанием событий перечислены все совпадающие события за указанный период времени. Нажмите на карточки событий, чтобы просмотреть полные детали события. Каждая карточка отображает:

- **Тип ресурса**: Тип ресурса Kubernetes, обозначенный иконками и сокращениями:
  - `P` = Pod
  - `RS` = ReplicaSet
  - `D` = Deployment
  - `SVC` = Service
- **Имя ресурса**: Название целевого ресурса.
- **Причина события**: Причина, зарегистрированная Kubernetes (например, FailedScheduling).
- **Уровень события**: Важность события.
  - `Normal` : Информационное
  - `Warning` : Требует немедленного внимания
- **Время**: Время последнего возникновения, количество появлений.

> **INFO**
>
> Kubernetes позволяет администраторам настраивать период хранения событий через контроллер Event TTL с периодом хранения по умолчанию 1 час. Просроченные события автоматически удаляются системой. Для получения полного исторического архива обращайтесь к разделу All Events.

Как сделать

Настройка правил срабатывания запланированных задач

Правила срабатывания запланированных задач поддерживают ввод выражений Crontab.

Содержание

Преобразование времени

Правило преобразования времени: Местное время - смещение часового пояса = UTC

В качестве примера возьмём **пекинское время и время UTC**:

Пекин находится в часовом поясе Восточного восьмого, разница между пекинским временем и UTC составляет 8 часов. Правило преобразования:

```
Пекинское время - 8 = UTC
```

Пример 1: пекинское время 9:42 преобразуется в UTC: 42 09 - 00 08 = 42 01, что означает время UTC 1:42 ночи.

Пример 2: пекинское время 4:32 утра преобразуется в UTC: 32 04 - 00 08 = -68 03. Если результат отрицательный, это означает предыдущий день, требуется дополнительное

преобразование: -68 03 + 00 24 = 32 20, что означает время UTC 20:32 предыдущего дня.

Запись выражений Crontab

Базовый формат и диапазон значений Crontab: `minute hour day month weekday`, с соответствующими диапазонами значений, приведёнными в таблице ниже:

Минута	Час	День	Месяц	День недели
[0-59]	[0-23]	[1-31]	[1-12] или [JAN-DEC]	[0-6] или [SUN-SAT]

Специальные символы, разрешённые в полях `minute hour day month weekday`, включают:

- `,`: Разделитель списка значений, используется для указания нескольких значений. Например: `1,2,5,7,8,9`.
- `-`: Пользовательский диапазон значений. Например: `2-4`, что означает 2, 3, 4.
- `*`: Представляет весь период времени. Например, для минут означает каждую минуту.
- `/`: Используется для указания шага увеличения значений. Например: `n/m` означает начиная с n, увеличивая на m каждый раз.

Ссылка на инструмент преобразования ↗

Распространённые примеры:

- Ввод `30 18 25 12 *` означает, что задача срабатывает в `18:30:00 25 декабря`.
- Ввод `30 18 25 * 6` означает, что задача срабатывает в `18:30:00 каждую субботу`.
- Ввод `30 18 * * 6` означает, что задача срабатывает в `18:30:00 по субботам`.
- Ввод `* 18 * * *` означает, что задача срабатывает каждую минуту начиная с `18:00:00` (включая `18:00:00`).
- Ввод `0 18 1,10,22 * *` означает, что задача срабатывает в `18:00:00` 1-го, 10-го и 22-го числа каждого месяца.

- Ввод `0,30 18-23 * * *` означает, что задача срабатывает в 00 и 30 минут каждого часа с 18:00 до 23:00 ежедневно.
- Ввод `* */1 * * *` означает, что задача срабатывает каждую минуту.
- Ввод `* 2-7/1 * * *` означает, что задача срабатывает каждую минуту с 2 до 7 часов утра ежедневно.
- Ввод `0 11 4 * mon-wed` означает, что задача срабатывает в `11:00` 4-го числа каждого месяца и по понедельникам, вторникам и средам.

Добавление ImagePullSecrets в ServiceAccount

Если для репозитория образов требуется аутентификация, необходимо добавить соответствующий `ImagePullSecrets` в `ServiceAccount`, используемый приложением. Это гарантирует, что приложение сможет успешно загружать образы из приватного репозитория.

Содержание

Создание ImagePullSecret

Для создания ImagePullSecret обратитесь к разделу Creating a Secret, где подробно описаны шаги по созданию ImagePullSecret.

Добавление ImagePullSecret в ServiceAccount

Если Pod вашего приложения использует `ServiceAccount` с именем `example`, вы можете добавить `ImagePullSecret` в `ServiceAccount` `example` в том namespace, где находится ваше приложение.

Отредактируйте `ServiceAccount` `example` с помощью команды patch:

```
kubectl patch serviceaccount example -p '{"imagePullSecrets": [{"name":
"my-registry-creds"}]}' -n <namespace>
```

Замените `<namespace>` на namespace, в котором находится ваше приложение, а `my-registry-creds` — на имя созданного вами `ImagePullSecret`.

Вы можете проверить добавление `ImagePullSecret`, описав `ServiceAccount`:

```
kubectl describe serviceaccount example -n <namespace>
Name:                example
Namespace:           <namespace>
Labels:              <none>
Annotations:         <none>
Image pull secrets:  my-registry-creds
Mountable secrets:   <none>
Tokens:              <none>
Events:              <none>
```

В разделе `Image pull secrets` вы должны увидеть добавленный секрет.

> **NOTE**
>
> **Примечание**: Если в вашем Pod не указан `ServiceAccount`, по умолчанию будет использоваться `default` `ServiceAccount` в namespace. Вы можете добавить `ImagePullSecret` в `default` `ServiceAccount` аналогичным образом.

Проверка установки imagePullSecrets для новых Pod

При создании нового Pod, использующего `ServiceAccount` `example`, Pod автоматически применит `ImagePullSecrets`, указанные в этом `ServiceAccount`.

Проверить это можно с помощью команды:

```
kubectl get pod <pod-name> -n <namespace> -o=jsonpath='{.spec.imagePullSecrets}'
```

Образы

Обзор образов

Обзор образов

Понимание контейнеров и образов

Образы

Реестр образов

Репозиторий образов

Теги образов

Идентификаторы образов

Контейнеры

Как сделать

Создание образов

Изучение лучших практик контейнеров

Включение метаданных в образы

Управление образами

Обзор политики загрузки образов

Разрешение pod использовать образы из других защищен

Создание секрета загрузки

Использование секрета загрузки в рабочей нагрузке

Обзор образов

Содержание

Понимание контейнеров и образов

Контейнеры и образы — важные понятия, которые необходимо понимать при создании и управлении контейнеризованным программным обеспечением. Образ содержит набор программного обеспечения, готового к запуску, тогда как контейнер — это запущенный экземпляр образа контейнера. Различные версии представлены разными тегами под одним и тем же именем образа.

Образы

Контейнеры в Alauda Container Platform основаны на образах контейнеров формата OCI. Образ — это бинарный файл, включающий все необходимые компоненты для запуска

одного контейнера, а также метаданные, описывающие его требования и возможности.

Можно рассматривать это как технологию упаковки. Контейнеры имеют доступ только к ресурсам, определённым в образе, если не предоставлен дополнительный доступ при создании. Развёртывая один и тот же образ в нескольких контейнерах на разных хостах и балансируя нагрузку между ними, Alauda Container Platform обеспечивает отказоустойчивость и горизонтальное масштабирование сервиса, упакованного в образ.

Вы можете использовать CLI `nerdctl` или `container` напрямую для сборки образов, но Alauda Container Platform также предоставляет builder-образы, которые помогают создавать новые образы, добавляя ваш код или конфигурацию к существующим образам.

Поскольку приложения развиваются со временем, одно имя образа может фактически ссылаться на множество различных версий одного и того же образа. Каждая версия образа уникально идентифицируется по его хешу — длинному шестнадцатеричному числу, например fd44297e2ddb050ec4f…, которое обычно сокращают до 12 символов, например fd44297e2ddb.

Реестр образов

Реестр образов — это сервер контента, который может хранить и предоставлять образы контейнеров. Например:

- Quay.io Container Registry ↗
- Alauda Container Platform Registry

Реестр содержит коллекцию одного или нескольких репозиториев образов, которые содержат один или несколько тегированных образов. Alauda Container Platform может предоставлять собственный реестр образов для управления пользовательскими образами контейнеров.

Репозиторий образов

Репозиторий образов — это коллекция связанных образов контейнеров и тегов, идентифицирующих их. Например, образы Jenkins для Alauda Container Platform

находятся в репозитории:

```
jenkins-2-centos7
```

Теги образов

Тег образа — это метка, применяемая к образу контейнера в репозитории, которая отличает конкретный образ от других образов в потоке образов. Обычно тег представляет собой номер версии. Например, здесь :v3 .11.59-2 — это тег:

```
jenkins-2-centos7:v3.11.59-2
```

Вы можете добавить дополнительные теги к образу. Например, образ может иметь теги :v3 .11.59-2 и :latest .

Идентификаторы образов

Идентификатор образа — это код SHA (Secure Hash Algorithm), который можно использовать для загрузки образа. SHA-идентификатор образа не может изменяться. Конкретный SHA всегда ссылается на точно такой же содержимое образа контейнера. Например:

```
jenkins-2-centos7@sha256:ab312bda324
```

Контейнеры

Основными единицами приложений в Alauda Container Platform являются контейнеры. Технологии контейнеров Linux — это лёгкие механизмы изоляции запущенных процессов, ограничивающие их взаимодействие только с назначенными ресурсами. Термин контейнер определяется как конкретный запущенный или приостановленный экземпляр образа контейнера.

Множество экземпляров приложений могут работать в контейнерах на одном хосте без видимости процессов, файлов, сети и т. д. друг друга. Обычно каждый контейнер предоставляет одну службу, часто называемую микросервисом, например веб-сервер или базу данных, хотя контейнеры могут использоваться для любых нагрузок.

Ядро Linux уже много лет включает возможности для технологий контейнеров. Ранние проекты runtime-контейнеров ввели удобные интерфейсы управления для запуска изолированных приложений на хосте. В последнее время Open Container Initiative ↗ разработала открытые стандарты для форматов контейнеров и runtime-контейнеров. Alauda Container Platform и Kubernetes добавляют возможность оркестрации контейнеров формата OCI в многохостовых установках.

Хотя при использовании Alauda Container Platform вы не взаимодействуете напрямую с runtime-контейнерами, понимание их возможностей и терминологии важно для понимания их роли в Alauda Container Platform и того, как ваши приложения функционируют внутри контейнеров.

Как сделать

Создание образов

Изучение лучших практик контейнеров

Включение метаданных в образы

Управление образами

Обзор политики загрузки образов

Разрешение pod использовать образы из других защищен

Создание секрета загрузки

Использование секрета загрузки в рабочей нагрузке

Создание образов

Узнайте, как создавать собственные контейнерные образы на основе предварительно собранных образов, которые готовы помочь вам. Процесс включает изучение лучших практик написания образов, определение метаданных для образов, тестирование образов и использование пользовательского рабочего процесса сборки для создания образов, которые можно использовать с Alauda Container Platform Registry. После создания образа вы можете отправить его в **Alauda Container Platform Registry**.

Содержание

Изучение лучших практик контейнеров

При создании контейнерных образов для запуска на Alauda Container Platform автору образа следует учитывать ряд лучших практик, чтобы обеспечить хороший опыт для пользователей этих образов. Поскольку образы предназначены быть неизменяемыми и использоваться как есть, следующие рекомендации помогают гарантировать, что ваши образы будут легко использоваться и хорошо восприниматься на Alauda Container Platform.

Общие рекомендации по контейнерным образам

Следующие рекомендации применимы при создании контейнерного образа в целом и не зависят от того, используются ли образы на Alauda Container Platform.

Повторное использование образов

По возможности базируйте свой образ на подходящем исходном образе, используя инструкцию FROM. Это гарантирует, что ваш образ сможет легко получать исправления безопасности из исходного образа при его обновлении, вместо того чтобы вам самим обновлять зависимости напрямую.

Кроме того, используйте теги в инструкции FROM, например, `alpine:3.20`, чтобы пользователи точно знали, на какой версии образа основан ваш образ. Использование тега, отличного от latest, гарантирует, что ваш образ не подвергнется несовместимым изменениям, которые могут появиться в последней версии исходного образа.

Поддерживайте совместимость внутри тегов

При тегировании собственных образов старайтесь сохранять обратную совместимость внутри одного тега. Например, если вы предоставляете образ с именем image, который в настоящее время включает версию `1.0`, вы можете использовать тег `image:v1`. При обновлении образа, если он остается совместимым с исходным, вы можете продолжать использовать тег `image:v1`, и потребители этого тега смогут получать обновления без сбоев.

Если позже вы выпускаете несовместимое обновление, переключитесь на новый тег, например `image:v2`. Это позволяет потребителям самостоятельно переходить на новую версию, не подвергаясь неожиданным сбоям из-за несовместимых изменений. Любой потребитель, использующий `image:latest`, принимает на себя риск любых несовместимых изменений.

Избегайте запуска нескольких процессов

Не запускайте несколько сервисов, таких как база данных и `SSHD`, внутри одного контейнера. Это не нужно, так как контейнеры легковесны и их легко связать вместе для оркестрации нескольких процессов. Alauda Container Platform позволяет легко размещать и совместно управлять связанными образами, группируя их в один pod.

Такое совместное размещение обеспечивает общий сетевой неймспейс и хранилище для связи между контейнерами. Обновления также менее разрушительны, так как каждый образ можно обновлять реже и независимо. Обработка сигналов также становится проще с одним процессом, так как не нужно управлять маршрутизацией сигналов к дочерним процессам.

Используйте exec в обёрточных скриптах

Многие образы используют обёрточные скрипты для настройки перед запуском основного процесса программного обеспечения. Если ваш образ использует такой скрипт, он должен использовать `exec`, чтобы процесс скрипта был заменён вашим программным процессом. Если не использовать `exec`, сигналы, отправленные вашим контейнерным рантаймом, будут направлены скрипту, а не процессу вашего ПО. Это нежелательно.

Например, если у вас есть обёрточный скрипт, который запускает серверный процесс. Вы запускаете контейнер, например, с помощью `podman run -i`, который запускает скрипт, а тот — ваш процесс. Если вы хотите закрыть контейнер с помощью `CTRL+C`, и если скрипт использовал `exec` для запуска сервера, `podman` отправит SIGINT серверному процессу, и всё будет работать как ожидается. Если `exec` не использовался, `podman` отправит SIGINT процессу скрипта, а ваш процесс продолжит работу, как будто ничего не произошло.

Также учтите, что ваш процесс работает как `PID 1` в контейнере. Это означает, что если основной процесс завершится, весь контейнер остановится, прерывая любые дочерние процессы, запущенные из процесса с `PID 1`.

Очистка временных файлов

Удаляйте все временные файлы, созданные во время сборки. Это также касается файлов, добавленных с помощью команды `ADD`. Например, выполняйте команду `yum clean` после операций `yum install`.

Вы можете предотвратить попадание кэша `yum` в слой образа, создавая команду `RUN` следующим образом:

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clea
n all -y
```

Обратите внимание, что если написать так:

```
RUN yum -y install mypackage
RUN yum -y install myotherpackage && yum clean all -y
```

то первый вызов `yum` оставит лишние файлы в этом слое, и эти файлы не смогут быть удалены при последующем выполнении `yum clean`. Лишние файлы не видны в итоговом образе, но присутствуют в базовых слоях.

Текущий процесс сборки контейнеров не позволяет команде, выполненной в более позднем слое, уменьшить занимаемое пространство, если что-то было удалено в более раннем слое. Однако это может измениться в будущем. Это означает, что если вы выполняете команду `rm` в более позднем слое, хотя файлы скрываются, общий размер образа для загрузки не уменьшается. Поэтому, как и в примере с `yum clean`, лучше удалять файлы в той же команде, где они создаются, чтобы они не попадали в слой.

Кроме того, выполнение нескольких команд в одной инструкции `RUN` уменьшает количество слоев в образе, что улучшает время загрузки и распаковки.

Располагаете инструкции в правильном порядке

Сборщик контейнеров читает `Dockerfile` и выполняет инструкции сверху вниз. Каждая успешно выполненная инструкция создаёт слой, который может быть повторно использован при следующей сборке этого или другого образа. Очень важно размещать инструкции, которые редко меняются, в начале `Dockerfile`. Это обеспечивает быструю сборку при повторных запусках, так как кэш не инвалидируется изменениями в верхних слоях.

Например, если вы работаете с `Dockerfile`, который содержит команду `ADD` для установки файла, над которым вы ведёте итерации, и команду `RUN` для установки пакета через `yum`, лучше поместить команду `ADD` последней:

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

Так при каждом изменении `myfile` и повторной сборке `podman build` система повторно использует кэш слоя для команды `yum` и создаёт новый слой только для операции `ADD`.

Если же написать `Dockerfile` так:

```
FROM foo
ADD myfile /test/myfile
RUN yum -y install mypackage && yum clean all -y
```

то при каждом изменении `myfile` и повторной сборке `podman build` операция `ADD` инвалидирует кэш слоя `RUN`, и команда `yum` будет выполняться заново.

Отмечайте важные порты

Инструкция `EXPOSE` делает порт в контейнере доступным для хоста и других контейнеров. Хотя можно указать, что порт должен быть открыт при запуске с помощью `podman run`, использование инструкции `EXPOSE` в `Dockerfile` облегчает использование вашего образа как для людей, так и для программ, явно объявляя порты, необходимые вашему ПО:

- Открытые порты отображаются в выводе `podman ps` для контейнеров, созданных из вашего образа.
- Открытые порты присутствуют в метаданных образа, возвращаемых командой `podman inspect`.
- Открытые порты связываются при связывании одного контейнера с другим.

Устанавливайте переменные окружения

Рекомендуется устанавливать переменные окружения с помощью инструкции `ENV`. Например, можно указать версию вашего проекта. Это облегчает пользователям поиск версии без просмотра `Dockerfile`. Другой пример — указание пути в системе, который может использоваться другим процессом, например, `JAVA_HOME`.

Избегайте паролей по умолчанию

Избегайте установки паролей по умолчанию. Многие расширяют образ и забывают удалить или изменить пароль по умолчанию. Это может привести к проблемам

безопасности, если пользователь в продакшене получит известный пароль. Пароли лучше настраивать через переменные окружения.

Если вы всё же устанавливаете пароль по умолчанию, убедитесь, что при запуске контейнера выводится соответствующее предупреждение. В сообщении должно быть указано значение пароля по умолчанию и объяснение, как его изменить, например, какую переменную окружения установить.

Избегайте sshd

Лучше не запускать `sshd` в вашем образе. Для доступа к контейнерам на локальном хосте используйте `podman exec`. В кластере применяйте `kubectl exec` для доступа к контейнерам, управляемым Alauda Container Platform. Установка и запуск `sshd` в образе увеличивает поверхность атаки и усложняет обновления.

Используйте тома для постоянных данных

Образы используют тома для хранения постоянных данных. Таким образом, Alauda Container Platform монтирует сетевое хранилище на узел, где запущен контейнер, и если контейнер перемещается на другой узел, хранилище подключается к новому узлу. Использование тома для всех постоянных данных сохраняет содержимое даже при перезапуске или перемещении контейнера. Если ваш образ записывает данные в произвольные места внутри контейнера, эти данные не сохранятся.

Все данные, которые должны сохраняться после удаления контейнера, должны записываться в том. Контейнерные движки поддерживают флаг `readonly` для контейнеров, который можно использовать для строгого соблюдения практик, запрещающих запись данных во временное хранилище контейнера. Проектирование образа с учётом этой возможности облегчает её использование в будущем.

Явное определение томов в вашем `Dockerfile` облегчает пользователям понимание, какие тома необходимо определить при запуске вашего образа.

Дополнительную информацию о томах и их использовании в Alauda Container Platform смотрите в документации Kubernetes↗.

> **Примечание:**
> Даже при использовании постоянных томов каждый экземпляр вашего образа имеет свой собственный том, и файловая система не разделяется между экземплярами. Это

> означает, что том нельзя использовать для совместного хранения состояния в кластере.

Включение метаданных в образы

Определение метаданных образа помогает Alauda Container Platform лучше использовать ваши контейнерные образы, создавая лучший опыт для разработчиков, использующих ваш образ. Например, вы можете добавить метаданные с полезными описаниями образа или предложениями других образов, которые могут понадобиться.

В этой теме определены только метаданные, необходимые для текущего набора сценариев использования. В будущем могут быть добавлены дополнительные метаданные или сценарии.

Определение метаданных образа

Вы можете использовать инструкцию `LABEL` в `Dockerfile` для определения метаданных образа. Метки похожи на переменные окружения тем, что представляют собой пары ключ-значение, прикреплённые к образу или контейнеру. Метки отличаются от переменных окружения тем, что они не видны запущенному приложению и могут использоваться для быстрого поиска образов и контейнеров.

Подробнее об инструкции `LABEL` смотрите в справочнике Dockerfile↗.

Имена меток обычно имеют пространство имён. Пространство имён устанавливается в соответствии с проектом, который будет использовать метки. Для Kubernetes пространство имён — io.k8s.

Управление образами

С помощью Alauda Container Platform вы можете взаимодействовать с образами в зависимости от того, где расположены реестры образов, какие требования к аутентификации существуют для этих реестров и как вы хотите, чтобы выполнялись ваши сборки и развертывания.

Политика загрузки образов

Каждый контейнер в pod использует образ контейнера. После того как вы создали образ и отправили его в реестр, вы можете ссылаться на него в pod.

Содержание

Обзор политики загрузки образов

Когда Alauda Container Platform создает контейнеры, она использует параметр `imagePullPolicy` контейнера, чтобы определить, нужно ли загружать образ перед запуском контейнера. Существует три возможных значения для `imagePullPolicy`:

Таблица значений `imagePullPolicy`:

Значение	Описание
Always	Всегда загружать образ.
IfNotPresent	Загружать образ только если он отсутствует на узле.
Never	Никогда не загружать образ.

Если параметр imagePullPolicy для контейнера не указан, Alauda Container Platform устанавливает его в зависимости от тега образа:

1. Если тег — latest, по умолчанию устанавливается imagePullPolicy со значением Always.
2. В противном случае по умолчанию устанавливается imagePullPolicy со значением IfNotPresent.

Использование секретов для загрузки образов

Если вы используете реестр образов Alauda Container Platform, то учетная запись сервиса вашего pod уже должна иметь необходимые разрешения, и дополнительных действий не требуется.

Однако в других сценариях, например при обращении к образам из разных проектов Alauda Container Platform или из защищенных реестров, требуется дополнительная настройка.

Разрешение pod использовать образы из других защищенных реестров

Чтобы загрузить защищенный контейнер из других приватных или защищенных реестров, необходимо создать секрет загрузки (pull secret) из учетных данных вашего

контейнерного клиента, например `Podman`, и добавить его в вашу учетную запись сервиса.

Контейнерные клиенты используют конфигурационный файл для хранения данных аутентификации для входа в защищенный или незащищенный реестр:

Эти файлы сохраняют ваши данные аутентификации, если вы ранее входили в защищенный или незащищенный реестр.

Создание секрета загрузки

Вы можете получить секрет загрузки образа для загрузки образа из приватного реестра контейнеров или репозитория. Подробнее см. Pull an Image from a Private Registry ↗.

Использование секрета загрузки в рабочей нагрузке

Вы можете использовать секрет загрузки, чтобы разрешить рабочим нагрузкам загружать образы из приватного реестра одним из следующих способов:

- Связать секрет с `ServiceAccount`, что автоматически применит секрет ко всем pod, использующим эту учетную запись сервиса.
- Определить `imagePullSecrets` в спецификации pod, что удобно для таких сред, как GitOps или ArgoCD.

Вы можете использовать секрет для загрузки образов для pod, добавив секрет в вашу учетную запись сервиса. Обратите внимание, что имя учетной записи сервиса должно совпадать с именем учетной записи сервиса, которую использует pod.

Пример вывода:

```
apiVersion: v1
imagePullSecrets:
- name: default-cfg-123456
- name: <pull_secret_name>
kind: ServiceAccount
metadata:
  name: default
  namespace: default
secrets:
- name: <pull_secret_name>
```

Вместо связывания секрета с учетной записью сервиса вы можете ссылаться на него напрямую в определении pod или рабочей нагрузки. Это удобно для рабочих процессов GitOps, таких как ArgoCD. Например:

Пример спецификации pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: <secure_pod_name>
spec:
  containers:
  - name: <container_name>
    image: your.registry.io/my-private-image
  imagePullSecrets:
  - name: <pull_secret_name>
```

Пример рабочего процесса ArgoCD:

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: <example_workflow>
spec:
  entrypoint: <main_task>
  imagePullSecrets:
  - name: <pull_secret_name>
```

Реестр

Введение

Введение

Принципы и изоляция по namespace

Аутентификация и авторизация

Преимущества

Сценарии применения

Установка

Установка через YAML

Когда использовать этот метод?

Предварительные требования

Установка Alauda Container Platform Reg

Обновление/Удаление Alauda Container

Установка через веб-интерфейс

Когда использовать этот метод?

Предварительные требования

Установка плагина кластера Alauda Container Platform Reg

Обновление/Удаление Alauda Container Platform Registry

Как сделать

Common CLI Command Operat

- Logging in Registry
- Add namespace permissions for users
- Add namespace permissions for a service
- Pulling Images
- Pushing Images

Using Alauda Container Platfor

- Рекомендации по доступу к реестру
- Развертывание примерного приложения
- Доступ между namespace
- Лучшие практики
- Контрольный список проверки
- Устранение неполадок

Резервное Platform Re

- Overview
- Prerequisites
- Data Backup
- Data Recovery
- Verification
- Solution Genera

Обновление

Руководство по обновлению реестра Alauda Container Platform

- Предварительные требования
- Обзор
- Шаги обновления

Введение

Создание, хранение и управление контейнерными образами является ключевой частью процесса разработки облачно-нативных приложений. Alauda Container Platform (ACP) предоставляет высокопроизводительный, высокодоступный встроенный сервис репозитория контейнерных образов, предназначенный для обеспечения пользователям безопасного и удобного опыта хранения и управления образами, значительно упрощая процессы разработки приложений, непрерывной интеграции/непрерывного развертывания (CI/CD) и развертывания приложений внутри платформы.

Глубоко интегрированный в архитектуру платформы, Alauda Container Platform Registry обеспечивает более тесное взаимодействие с платформой, упрощённую конфигурацию и большую эффективность внутреннего доступа по сравнению с внешним, независимо развернутым репозиторием образов.

Содержание

Принципы и изоляция по namespace

Встроенный репозиторий образов Alauda Container Platform, как один из ключевых компонентов платформы, работает внутри кластера в режиме высокой доступности и использует возможности постоянного хранения, предоставляемые платформой, чтобы гарантировать безопасность и надёжность данных образов.

Одним из основных концептов его проектирования является логическая изоляция и управление на основе Namespace. Внутри Registry репозитории образов организованы по namespace. Это означает, что каждый namespace можно рассматривать как отдельную «зону» для образов, принадлежащих этому namespace, и образы между разными namespace по умолчанию изолированы, если только явно не предоставлены права доступа.

Аутентификация и авторизация

Механизм аутентификации и авторизации Alauda Container Platform Registry глубоко интегрирован с системой аутентификации и авторизации уровня платформы ACP, обеспечивая контроль доступа с точностью до namespace:

Аутентификация

Пользователям или автоматизированным процессам (например, CI/CD пайплайнам на платформе, автоматическим задачам сборки и т.п.) не нужно поддерживать отдельный набор учётных данных для Registry. Аутентификация происходит через стандартные механизмы платформы (например, с использованием предоставляемых платформой API токенов, интегрированных корпоративных систем идентификации и т.д.). При доступе к Alauda Container Platform Registry через CLI или другие инструменты обычно используется существующая сессия входа на платформу или токены ServiceAccount для прозрачной аутентификации.

Авторизация

Контроль авторизации реализован на уровне namespace. Права Pull или Push для репозитория образов в Alauda Container Platform Registry зависят от роли и разрешений пользователя или ServiceAccount в соответствующем namespace.

- **Как правило**, владельцу или роли разработчика namespace автоматически предоставляются права Push и Pull для репозиториев образов в этом namespace.
- **Пользователи из других namespace или желающие выполнять pull образов между namespace** должны получить явное разрешение от администратора целевого namespace (например, привязка роли с правом pull через RBAC) перед тем, как получить доступ к образам в этом namespace.
- **Данный механизм авторизации на основе namespace** обеспечивает изоляцию образов между namespace, повышая безопасность и предотвращая несанкционированный доступ и изменение.

Преимущества

Основные преимущества Alauda Container Platform Registry:

- **Готовность к использованию:** Быстрое развертывание приватного реестра образов без сложных настроек.
- **Гибкий доступ:** Поддержка как внутрикластерного, так и внешнего доступа.
- **Гарантия безопасности:** Обеспечение авторизации RBAC и возможностей сканирования образов.
- **Высокая доступность:** Обеспечение непрерывности сервиса через механизмы репликации.
- **Промышленный уровень:** Проверено в корпоративных средах с гарантией SLA.

Сценарии применения

- **Лёгкое развертывание:** Реализация упрощённых решений реестра в средах с низкой нагрузкой для ускорения доставки приложений.
- **Edge Computing:** Обеспечение автономного управления для edge-кластеров с выделенными реестрами.
- **Оптимизация ресурсов:** Демонстрация возможностей полного рабочего процесса через интегрированные решения Source to Image (S2I) при недостаточном использовании инфраструктуры.

Установка

Установка через YAML

Содержание

Когда использовать этот метод?

Рекомендуется для:

- **Продвинутых пользователей** с опытом работы с Kubernetes, предпочитающих ручной подход.
- **Производственных развертываний**, требующих корпоративного хранилища (NAS, AWS S3, Ceph и др.).
- Сред, где необходим **тонкий контроль** над TLS, ingress.

- **Полной настройки YAML** для сложных конфигураций.

Предварительные требования

- **Установить** плагин кластера **Alauda Container Platform Registry** в целевой кластер.
- **Доступ** к целевому Kubernetes кластеру с настроенным kubectl.
- **Права администратора кластера** для создания ресурсов с областью действия кластера.
- Получить зарегистрированный **домен** (например, registry.yourcompany.com) Create a Domain
- Обеспечить действующее **NAS-хранилище** (например, NFS, GlusterFS и др.).
- (Опционально) Обеспечить действующее **S3-хранилище** (например, AWS S3, Ceph и др.). Если S3-хранилище отсутствует, разверните MinIO (встроенный S3) в кластере Deploy MinIO.

Установка Alauda Container Platform Registry через YAML

Процедура

1. **Создайте YAML-файл конфигурации** с именем registry-plugin.yaml по следующему шаблону:

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ClusterPluginInstance
metadata:
  annotations:
    cpaas.io/display-name: image-registry
  labels:
    create-by: cluster-transformer
    manage-delete-by: cluster-transformer
    manage-update-by: cluster-transformer
  name: image-registry
spec:
  config:
    access:
      address: ''
      enabled: false
    fake:
      replicas: 2
    infra:
      enabled: false
    global:
      expose: false
      isIPv6: false
      replicas: 2
      oidc:
        ldapID: ''
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 250m
          memory: 256Mi
    ingress:
      enabled: true
      hosts:
        - name: <YOUR-DOMAIN> # [REQUIRED] Настройте домен
          tlsCert: <NAMESPACE>/<TLS-SECRET> # [REQUIRED] Namespace/Secr
etName
      ingressClassName: '<INGRESS-CLASS-NAME>' # [REQUIRED] IngressClas
sName
      insecure: false
    persistence:
      accessMode: ReadWriteMany
```

```
      nodes: ''
      path: <YOUR-HOSTPATH> # [REQUIRED] Локальный путь для LocalVolume
      size: <STORAGE-SIZE> # [REQUIRED] Размер хранилища (например, 10G
i)
      storageClass: <STORAGE-CLASS-NAME> # [REQUIRED] Имя StorageClass
      type: StorageClass
    s3storage:
      bucket: <S3-BUCKET-NAME> # [REQUIRED] Имя S3 bucket
      enabled: false # Установите false для локального хранилища
      env:
        REGISTRY_STORAGE_S3_SKIPVERIFY: false # Установите true для сам
оподписанных сертификатов
      region: <S3-REGION> # Регион S3
      regionEndpoint: <S3-ENDPOINT> # Endpoint S3
      secretName: <S3-CREDENTIALS-SECRET> # Секрет с учетными данными S
3
    service:
      nodePort: ''
      type: ClusterIP
  pluginName: image-registry
```

2. **Настройте следующие поля** в соответствии с вашей средой:

```
spec:
  config:
    global:
      oidc:
        ldapID: '<LDAP-ID>' # LDAP ID
    infra:
      enabled: false  # Если хотите развернуть компоненты на infra-нода
x. По умолчанию false — все ноды.
    ingress:
      hosts:
        - name: '<YOUR-DOMAIN>' # например, registry.your-company.com
          tlsCert: '<NAMESPACE>/<TLS-SECRET>' # например, cpaas-system/
tls-secret
      ingressClassName: '<INGRESS-CLASS-NAME>' # например, cluster-alb-
1
    persistence:
      size: '<STORAGE-SIZE>' # например, 10Gi
      storageClass: '<STORAGE-CLASS-NAME>' # например, cpaas-system-sto
rage
    s3storage:
      bucket: '<S3-BUCKET-NAME>' # например, prod-registry
      region: '<S3-REGION>' # например, us-west-1
      regionEndpoint: '<S3-ENDPOINT>' # например, https://s3.amazonaws.
com
      secretName: '<S3-CREDENTIALS-SECRET>' # Секрет с AWS_ACCESS_KEY_I
D/AWS_SECRET_ACCESS_KEY
      env:
        REGISTRY_STORAGE_S3_SKIPVERIFY: 'true' # Установите "true" для
самоподписанных сертификатов
```

3. **Как создать секрет** для учетных данных S3:

```
kubectl create secret generic <S3-CREDENTIALS-SECRET> \
  --from-literal=access-key-id=<YOUR-S3-ACCESS-KEY-ID> \
  --from-literal=secret-access-key=<YOUR-S3-SECRET-ACCESS-KEY> \
  -n cpaas-system
```

   Замените `<S3-CREDENTIALS-SECRET>` на имя вашего секрета с учетными данными S3.

4. Примените конфигурацию в ваш кластер:

```
kubectl apply -f registry-plugin.yaml
```

Справочник по конфигурации

Обязательные поля

Параметр	Описание	Пр
`spec.config.global.oidc.ldapID`	LDAP ID для аутентификации OIDC	`l`
`spec.config.ingress.hosts[0].name`	Пользовательский домен для доступа к registry	`re`
`spec.config.ingress.hosts[0].tlsCert`	Ссылка на секрет TLS сертификата (namespace/secret-name)	`cp` `tls`
`spec.config.ingress.ingressClassName`	Имя класса ingress для registry	`c`
`spec.config.persistence.size`	Размер хранилища для registry	`10`
`spec.config.persistence.storageClass`	Имя StorageClass для registry	`nf`
`spec.config.s3storage.bucket`	Имя S3 bucket для хранения образов	`pr`
`spec.config.s3storage.region`	Регион AWS для S3 хранилища	`us`
`spec.config.s3storage.regionEndpoint`	URL endpoint сервиса S3	`ht`

Параметр	Описание	Пр
`spec.config.s3storage.secretName`	Секрет с учетными данными S3	`s3`
`spec.config.s3storage.env.REGISTRY_STORAGE_S3_SKIPVERIFY`	Установите `true` для самоподписанных сертификатов	`tr`
`spec.config.infra.enabled`	Развернуть компоненты на infra-нодах или на всех нодах	`fa`

Проверка

1. Проверить плагин:

```
kubectl get clusterplugininstances image-registry -o yaml
```

2. Проверить поды registry:

```
kubectl get pods -n cpaas-system -l app=image-registry
```

Обновление/Удаление Alauda Container Platform Registry

Обновление

Выполните следующую команду в глобальном кластере и обновите значения в ресурсе согласно описанию параметров выше для завершения обновления:

```
# <CLUSTER-NAME> — кластер, где установлен плагин
kubectl edit -n cpaas-system \
  $(kubectl get moduleinfo -n cpaas-system -l cpaas.io/cluster-name=<CLUS
TER-NAME>,cpaas.io/module-name=image-registry -o name)
```

Удаление

Выполните следующую команду в глобальном кластере:

```
# <CLUSTER-NAME> — кластер, где установлен плагин
kubectl get moduleinfo -n cpaas-system -l cpaas.io/cluster-name=<CLUSTER-
NAME>,cpaas.io/module-name=image-registry -o name | xargs kubectl delete
-n cpaas-system
```

Установка через веб-интерфейс

Содержание

Когда использовать этот метод?

Рекомендуется для:

- **Начинающих пользователей**, предпочитающих пошаговый визуальный интерфейс.
- **Быстрой настройки proof-of-concept** в непроизводственных средах.
- Команд с **ограниченными знаниями Kubernetes**, которым нужен упрощённый процесс развертывания.
- Сценариев, требующих **минимальной кастомизации** (например, стандартные настройки хранилища).
- **Базовых сетевых конфигураций** без специфичных правил ingress.
- Настроек **StorageClass** для обеспечения высокой доступности.

Не рекомендуется для:

- Производственных сред, требующих продвинутых настроек хранилища (S3 storage).
- Сетевых конфигураций с необходимостью специфичных правил ingress.

Предварительные требования

- **Установите** плагин кластера **Alauda Container Platform Registry** в целевой кластер с помощью механизма Cluster Plugin.

Установка плагина кластера Alauda Container Platform Registry через веб-консоль

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 **Alauda Container Platform Registry**, нажмите **Install** и перейдите на страницу установки.
4. Настройте параметры согласно следующим спецификациям и нажмите **Install** для завершения развертывания.

Описание параметров:

Параметр	Описание
Expose Service	При включении администраторы смогут управлять репозиторием образов извне по адресу доступа. Это представляет значительные риски безопасности и должно включаться с крайней осторожностью.
Enable IPv6	Включите эту опцию, если кластер использует одностековую сеть IPv6.

Параметр	Описание
NodePort	При включённом Expose Service настройте NodePort для обеспечения внешнего доступа к Registry через этот порт.
Storage Type	Выберите тип хранилища. Поддерживаемые типы: LocalVolume и StorageClass.
Nodes	Выберите узел для запуска сервиса Registry для хранения и распространения образов. (Доступно только при выборе Storage Type = LocalVolume)
StorageClass	Выберите StorageClass. При количестве реплик более 1 выбирайте хранилище с поддержкой RWX (ReadWriteMany) (например, File Storage) для обеспечения высокой доступности. (Доступно только при StorageClass)
Storage Size	Объём хранилища, выделенный для Registry (единица измерения: Gi).
Replicas	Настройте количество реплик Pod Registry: • **LocalVolume**: по умолчанию 1 (фиксировано) • **StorageClass**: по умолчанию 3 (можно изменять)
Resource Requirements	Определите запросы и лимиты по CPU и памяти для Pod Registry.

Проверка

1. Перейдите в **Marketplace** > **Cluster Plugins** и убедитесь, что статус плагина отображается как **Installed**.
2. Нажмите на название плагина для просмотра его деталей.
3. Скопируйте **Registry Address** и используйте клиент контейнеров (например, Podman) для загрузки/выгрузки образов.

Обновление/Удаление Alauda Container Platform Registry

Вы можете обновить или удалить плагин **Alauda Container Platform Registry** как со страницы списка, так и со страницы деталей.

Как сделать

Common CLI Command Operat

- Logging in Registry
- Add namespace permissions for users
- Add namespace permissions for a service
- Pulling Images
- Pushing Images

Using Alauda Container Platfor

- Рекомендации по доступу к реестру
- Развертывание примерного приложения
- Доступ между namespace
- Лучшие практики
- Контрольный список проверки
- Устранение неполадок

Резервное Platform Re

- Overview
- Prerequisites
- Data Backup
- Data Recovery
- Verification
- Solution Genera

Common CLI Command Operations

Платформа Alauda Container Platform предоставляет инструменты командной строки для взаимодействия пользователей с реестром Alauda Container Platform. Ниже приведены примеры распространённых операций и команд:

Предположим, что адрес сервиса реестра для кластера — registry.cluster.local, а пространство имён, с которым вы работаете, — my-ns.

> Свяжитесь с технической службой, чтобы получить плагин kubectl-acp и убедиться, что он корректно установлен в вашей среде.

Содержание

Logging in Registry

Войдите в реестр кластера, выполнив вход в ACP.

```
kubectl acp login <ACP-endpoint>
```

Add namespace permissions for users

Добавьте пользователю разрешение на pull в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-pull
er --user=<username> -n <namespace>
```

Добавьте пользователю разрешение на push в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-push
er --user=<username> -n <namespace>
```

Add namespace permissions for a service account

Добавьте сервисному аккаунту разрешение на pull в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-pull
er --serviceaccount=<namespace>:<serviceaccount-name> -n <namespace>
```

Добавьте сервисному аккаунту разрешение на push в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-push
er --serviceaccount=<namespace>:<serviceaccount-name> -n <namespace>
```

Pulling Images

Загружает образ из реестра внутрь кластера (например, для развертывания Pod).

```
# Загрузить образ с именем my-app и тегом latest из реестра текущего прос
транства имён (my-ns)
kubectl acp pull registry.cluster.local/my-ns/my-app:latest

# Загрузить образы из других пространств имён (например, shared-ns) (треб
уется разрешение на pull из пространства shared-ns)
kubectl acp pull registry.cluster.local/shared-ns/base-image:latest
```

Эта команда проверяет вашу личность и права на pull в целевом пространстве имён, а затем загружает образ из реестра.

Pushing Images

Отправляет локально собранные образы или образы, загруженные из других источников, в определённое пространство имён реестра.

Сначала необходимо пометить (tag) локальный образ адресом и форматом пространства имён целевого реестра с помощью стандартного инструмента командной строки для контейнеров, например podman.

```
# Пометить образ целевым адресом:
podman tag my-app:latest registry.cluster.local/my-ns/my-app:v1

# Использовать команду kubectl для отправки образа в реестр текущего прос
транства имён (my-ns)
kubectl acp push registry.cluster.local/my-ns/my-app:v1
```

Отправляет образ из удалённого репозитория образов в определённое пространство имён реестра Alauda Container Platform.

```
# Если в вашем удалённом репозитории образов есть образ remote.registry.i
o/demo/my-app:latest
# Используйте команду kubectl для отправки его в пространство имён (my-n
s) реестра
kubectl acp push remote.registry.io/demo/my-app:latest registry.cluster.l
ocal/my-ns/my-app:latest
```

Эта команда проверяет вашу личность и права на push в пространстве имён my-ns, а затем загружает локально помеченный образ в реестр.

Using Alauda Container Platform Registry in Kubernetes Clusters

Реестр Alauda Container Platform (ACP) обеспечивает безопасное управление образами контейнеров для рабочих нагрузок Kubernetes.

Содержание

Рекомендации по доступу к реестру

- **Рекомендуется использовать внутренний адрес**: Для образов, хранящихся в реестре кластера, при развертывании внутри кластера всегда отдавайте предпочтение внутреннему сервисному адресу `image-registry.cpaas-system.svc`. Это обеспечивает оптимальную сетевую производительность и избегает ненужной маршрутизации через внешние сети.

- **Использование внешнего адреса**: Внешний ingress-домен (например, `registry.cluster.local`) предназначен в первую очередь для:
  - загрузки и выгрузки образов из вне кластера (например, с машин разработчиков, CI/CD систем)
  - операций вне кластера, требующих доступа к реестру

Развертывание примерного приложения

1. Создайте приложение с именем `my-app` в namespace `my-ns`.
2. Сохраните образ приложения в реестре по адресу `image-registry.cpaas-system.svc/my-ns/my-app:v1`.
3. **По умолчанию** ServiceAccount в каждом namespace автоматически настраивается с imagePullSecret для доступа к образам из `image-registry.cpaas-system.svc`.

Пример Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-ns
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: main-container
        image: image-registry.cpaas-system.svc/my-ns/my-app:v1
        ports:
        - containerPort: 8080
```

Доступ между namespace

Чтобы разрешить пользователям из `my-ns` вытягивать образы из `shared-ns`, администратор `shared-ns` может создать role binding, предоставляющий необходимые права.

Пример Role Binding

```
# Доступ к образам из другого namespace (требуются права)
kubectl create rolebinding cross-ns-pull \
  --clusterrole=system:image-puller \
  --serviceaccount=my-ns:default \
  -n shared-ns
```

Лучшие практики

- **Использование реестра**: Всегда используйте `image-registry.cpaas-system.svc` для развертываний, чтобы обеспечить безопасность и производительность.
- **Изоляция namespace**: Используйте изоляцию namespace для повышения безопасности и управления образами.
  - Используйте пути образов, основанные на namespace: `image-registry.cpaas-system.svc/<namespace>/<image>:<tag>`.
- **Управление доступом**: Используйте role binding для управления доступом между namespace для пользователей и сервисных аккаунтов.

Контрольный список проверки

1. Проверьте доступность образа для ServiceAccount по умолчанию в `my-ns`:

```
kubectl auth can-i get images.registry.alauda.io --namespace my-ns --as
=system:serviceaccount:my-ns:default
```

2. Проверьте доступность образа для пользователя в `my-ns`:

```
kubectl auth can-i get images.registry.alauda.io --namespace my-ns --as
=<username>
```

Устранение неполадок

- **Ошибки при загрузке образа**: Проверьте imagePullSecrets в спецификации pod и убедитесь, что они настроены корректно.
- **Отказ в доступе**: Убедитесь, что у пользователя или ServiceAccount есть необходимые role binding в целевом namespace.
- **Проблемы с сетью**: Проверьте сетевые политики и конфигурацию сервисов для обеспечения подключения к внутреннему реестру.
- **Сбои DNS**: Проверьте содержимое файла `/etc/hosts` на узле, убедитесь, что разрешение DNS для `image-registry.cpaas-system.svc` настроено правильно.
  - Проверьте конфигурацию /etc/hosts на узле для корректного разрешения DNS `image-registry.cpaas-system.svc`
  - Пример отображения сервиса реестра (ClusterIP сервиса image-registry):

    ```
    # /etc/hosts
    127.0.0.1   localhost localhost.localdomain
    10.4.216.11 image-registry.cpaas-system image-registry.cpaas-system.s
    vc image-registry.cpaas-system.svc.cluster.local # cpaas-generated-no
    de-resolver
    ```

  - **Как получить текущий ClusterIP `image-registry`**:

    ```
    kubectl get svc -n cpaas-system image-registry -o jsonpath='{.spec.cl
    usterIP}'
    ```

Резервное копирование и восстановление данных Alauda Container Platform Registry

Содержание

Overview

Данное решение предоставляет рекомендации по резервному копированию и восстановлению данных `Alauda Container Platform Registry`, использующего объектное хранилище, совместимое с S3, в Alauda Container Platform (ACP).

Основная идея: разделить управление самими данными образов (хранящимися в S3) и конфигурацией `cluster plugin` (определённой в пользовательском ресурсе Kubernetes `ModuleInfo`).

- **Резервное копирование**: получение конфигурации S3 из ресурса `ModuleInfo` и резервное копирование данных из указанного хранилища `bucket`.
- **Восстановление**: после установки `cluster plugin` в новом кластере обновить его конфигурацию S3 в ресурсе `ModuleInfo`, указав bucket с восстановленными данными, тем самым завершив интеграцию данных.

Преимущества:

- **Разделение операций**: резервное копирование и восстановление данных не зависят от процессов развертывания и обновления ACP `cluster plugin`.
- **Управление через конфигурацию**: вся информация о подключении управляется декларативным ресурсом `ModuleInfo`, что обеспечивает безопасные и надёжные изменения.
- **Расширяемость**: данный шаблон можно применить к другим типам хранилищ (например, локальная файловая система, StorageClass, NAS).

Prerequisites

- Наличие доступа `kubectl` и соответствующих прав для работы с целевым Kubernetes кластером.
- Наличие учётных данных и клиентских инструментов (например, awscli, rclone, minio-client) для доступа и работы с объектным хранилищем, совместимым с S3, используемым для хранения данных образов.
- Установленный и настроенный `Alauda Container Platform Registry` cluster plugin, а также существующий и работоспособный ресурс `ModuleInfo`.
- Подготовленное независимое и достаточное по объёму хранилище для резервных данных (например, другой S3 bucket).

Data Backup

Цель данного этапа — получить текущую рабочую конфигурацию S3 и выполнить полное резервное копирование данных образов из хранилища bucket.

Step 1: Obtain Current S3 Configuration

Извлеките конфигурацию хранилища S3 из ресурса `ModuleInfo`, управляющего `Alauda Container Platform Registry` cluster plugin. Эта информация является основой для операции резервного копирования.

Следующие команды следует выполнять в `global` кластере ACP:

```
# 1. Определить имя ресурса ModuleInfo для модуля image-registry
MODULE_INFO_NAME=$(kubectl get moduleinfo -l cpaas.io/module-name=image-registry -o jsonpath='{.items[0].metadata.name}')
echo "Target ModuleInfo Resource: $MODULE_INFO_NAME"

# 2. Извлечь ключевую информацию конфигурации S3
S3_BUCKET=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.bucket}')
S3_ENDPOINT=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.regionEndpoint}')
S3_REGION=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.region}')
S3_SECRET_NAME=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.secretName}')

# 3. Получить ключи доступа из Secret (обычно access-key-id и secret-access-key)
# Примечание: вывод закодирован в Base64 и требует соответствующего декодирования.
kubectl get secret -n cpaas-system $S3_SECRET_NAME -o jsonpath='{.data}'
```

Описание ключевых переменных:

- `S3_BUCKET`: имя исходного bucket, где фактически хранятся данные образов.
- `S3_ENDPOINT`: URL-адрес подключения к сервису, совместимому с S3.

- `S3_REGION`: идентификатор региона S3.
- `S3_SECRET_NAME`: имя Kubernetes Secret, содержащего ключи аутентификации.

Step 2: Perform S3 Bucket Data Backup

Используя выбранный клиент S3, примените полученную на предыдущем шаге конфигурацию для полного резервного копирования данных из исходного bucket.

Логика работы:

- Настройте клиент с использованием endpoint ($S3_ENDPOINT), региона ($S3_REGION) и декодированных ключей доступа из Secret.
- Выполните команду синхронизации или копирования для резервного копирования всех данных из исходного bucket ($S3_BUCKET) в подготовленное независимое место хранения (например, другой S3 bucket или путь).
- Зафиксируйте время резервного копирования, имя bucket и endpoint, используемые при операции, и сохраните эту информацию вместе с резервными файлами.

Data Recovery

На данном этапе предполагается, что `Alauda Container Platform Registry` cluster plugin успешно установлен в целевой среде (новом или восстановленном кластере) через платформу. Цель — изменить его конфигурацию для доступа к восстановленным данным образов.

Step 1: Prepare Backup Data

Используя выбранный клиент S3, восстановите резервные данные образов в **целевой S3 storage `bucket`, к которому гарантирован доступ**. Например, восстановите в новый bucket с именем `registry-bucket-restored`. Убедитесь, что у вас есть права на запись в этот bucket.

Step 2: Update ModuleInfo Configuration

Ключ к восстановлению — обновить ресурс `ModuleInfo` нового `cluster plugin`, указав в его конфигурации S3 целевой bucket с резервными данными.

1. **Определите новую информацию для подключения к S3**:

- `NEW_BUCKET`: **имя целевого bucket**, куда восстановлены данные (например, registry-bucket-restored).
- `NEW_ENDPOINT`: **endpoint целевого S3 сервиса**. Если адрес S3 не изменился, остаётся прежним.
- `NEW_REGION`: **регион целевого S3 сервиса**.
- `NEW_SECRET_NAME`: имя Kubernetes Secret с правами чтения/записи для **целевого bucket**. Если ключи доступа не изменились, это всё ещё `$S3_SECRET_NAME`.

2. **Обновите ресурс ModuleInfo**: Используйте команду kubectl patch для прямого обновления секции конфигурации S3 в `ModuleInfo`. Контроллер платформы автоматически синхронизирует это изменение со всеми соответствующими Deployment, Pod и другими ресурсами.

```
# Выполнить обновление конфигурации
kubectl patch moduleinfo $MODULE_INFO_NAME --type=merge -p '{
  "spec": {
    "config": {
      "s3storage": {
        "bucket": "'"$NEW_BUCKET"'",
        "regionEndpoint": "'"$NEW_ENDPOINT"'",
        "region": "'"$NEW_REGION"'",
        "secretName": "'"$NEW_SECRET_NAME"'"
      }
    }
  }
}'
```

Важный момент: эта операция запускает rolling update Pod-ов, связанных с `Alauda Container Platform Registry`. Новые Pod-ы будут использовать обновлённую конфигурацию для подключения к указанному целевому bucket.

Verification

После завершения обновления выполните следующие шаги для проверки успешного восстановления данных и нормальной работы сервиса.

Check Module Status(in global cluster)

```
# Проверить, что Pod-ы успешно перезапустились и работают с новой конфигурацией
kubectl get pods -n cpaas-system -l app=image-registry
# Просмотреть логи Pod-ов для подтверждения отсутствия ошибок подключения к S3
kubectl logs -n cpaas-system -l app=image-registry -c registry --tail=50
```

Verify Data Access (API Test)

Используйте API Registry для прямой проверки возможности чтения восстановленных данных образов.

```
# Получить адрес доступа к сервису Registry (предполагается тип ClusterIP)
REGISTRY_SVC_IP=$(kubectl get svc -n cpaas-system image-registry -o jsonpath='{.spec.clusterIP}')

# Тест 1: Запрос каталога репозиториев
curl -s http://$REGISTRY_SVC_IP/v2/_catalog | jq .
# Ожидаемый успешный ответ: {"repositories":["image1","image2",...]}

# Тест 2: Запрос списка тегов для конкретного образа (например, образа `myns/nginx`)
curl -s http://$REGISTRY_SVC_IP/v2/myns/nginx/tags/list | jq .
# Ожидаемый успешный ответ: {"name":"myns/nginx","tags":["v1.0","latest",...]}
```

Functionality Test

Попробуйте выполнить pull известного образа из восстановленного реестра или запушить новый образ, чтобы полностью проверить функциональность чтения и записи.

Solution Generality

Хотя в данном решении в качестве примера используется хранилище S3, его **архитектурный шаблон применим к различным типам хранилищ**, поддерживаемых Registry (например, локальная файловая система, StorageClass, NAS).

Общий принцип: независимо от типа хранилища, основной процесс резервного копирования и восстановления остаётся одинаковым. Сначала извлекаются параметры подключения к хранилищу из соответствующего блока конфигурации (например, `s3storage`, `persistence`) в ресурсе `ModuleInfo`, затем с помощью соответствующих инструментов выполняется резервное копирование данных. Для восстановления после восстановления данных в целевое место достаточно обновить соответствующие поля конфигурации в `ModuleInfo`. Платформа автоматически направит вновь развернутый экземпляр к этому месту.

Основная ценность: используя **единый уровень абстракции конфигурации** (`ModuleInfo`), данное решение отделяет процесс резервного копирования и восстановления данных от конкретных реализаций хранилищ и развертывания приложений в Kubernetes, обеспечивая стандартизированное управление и расширяемость.

Обновление

Руководство по обновлению реестра Alauda Container Platform

Руководство по обновлению реестра Alauda Container Platform

Содержание

Предварительные требования

1. Привилегии администратора для Alauda Container Platform.
2. Старый плагин относится к версиям Alauda Container Platform v4.1 (включительно) и ранее.
3. Новый плагин относится к версиям Alauda Container Platform v4.2 (включительно) и позже.

Обзор

В этом документе приведены инструкции по обновлению `Old Plugin` до `New Plugin`. Из-за изменения имени плагина кластера требуется ручное вмешательство в зависимости от **типа хранилища**.

Шаги обновления

Проверка окружения и предварительная подготовка

Старый плагин не установлен

Если старый плагин никогда не устанавливался, необходимо просто очистить старый плагин из [Marketplace/Cluster Plugins]:

```
# Выполнить в глобальном кластере
kubectl delete moduleplugins internal-docker-registry
kubectl get moduleconfig -l cpaas.io/module-name=internal-docker-registry
-o name | xargs kubectl delete
```

Старый плагин установлен

Если старый плагин установлен в любом кластере, следуйте процедуре миграции в зависимости от вашего **типа хранилища**.

Миграция для хранилища S3

Особенности: Автоматическая миграция данных, ручные операции не требуются

Процедура:

1. Резервное копирование конфигурации старого плагина: (Выполнить в рабочем кластере, где установлен старый плагин)

```
kubectl get clusterplugininstances internal-docker-registry -o yaml > b
ackup-clusterplugininstances.yaml
```

2. Удаление старого плагина: (Выполнить в глобальном кластере)

```
# Замените <CLUSTER-NAME> на фактическое имя кластера, в котором устано
влен плагин
kubectl get moduleinfo -l cpaas.io/cluster-name=<CLUSTER-NAME>,cpaas.i
o/module-name=internal-docker-registry -o name | xargs kubectl delete
```

3. Установка нового плагина (Выполнить в рабочем кластере, где будет установлен плагин)

- Отредактируйте файл резервной копии `backup-clusterplugininstances.yaml`, заменив все `internal-docker-registry` на `image-registry`
- Примените новую конфигурацию:

```
kubectl apply -f backup-clusterplugininstances.yaml
```

4. Очистка старого плагина из [Marketplace/Cluster Plugins]: (Выполнить в глобальном кластере)

```
kubectl delete moduleplugins internal-docker-registry
kubectl get moduleconfig -l cpaas.io/module-name=internal-docker-regist
ry -o name | xargs kubectl delete
```

Миграция для хранилища NFS

Особенности: Требуется ручное сохранение PV и изменение политики восстановления.

Процедура:

1. Запись информации о PV: (Выполнить в рабочем кластере, где установлен старый плагин)

```
OLD_PV=$(kubectl get pvc internal-docker-registry -n cpaas-system -o js
onpath='{.spec.volumeName}')
echo "Имя старого PV: $OLD_PV"
```

2. Изменение политики восстановления PV: (Выполнить в рабочем кластере, где установлен старый плагин)

```
kubectl patch pv $OLD_PV -p '{"spec":{"persistentVolumeReclaimPolic
y":"Retain"}}'
```

3. Резервное копирование конфигурации старого плагина (То же, что и шаг 1 для S3)
4. Удаление старого плагина (То же, что и шаг 2 для S3)
5. Освобождение привязки PV: (Выполнить в рабочем кластере, где установлен старый плагин)

```
kubectl patch pv $OLD_PV -p '{"spec":{"claimRef":null}}'
```

6. Установка нового плагина (Важно: необходимо установить `config.persistence.volumeName` для нового плагина, чтобы сохранить старые данные)

- Отредактируйте файл конфигурации `backup-clusterplugininstances.yaml`, установите config.persistence.volumeName: `<OLD_PV>`
- Примените конфигурацию (То же, что и шаг 3 для S3)

7. Очистка старого плагина из [Marketplace/Cluster Plugins] (То же, что и шаг 4 для S3)

Миграция для локального хранилища

Особенности: Требуется ручное копирование данных

Процедура:

1. Резервное копирование конфигурации старого плагина (То же, что и шаг 1 для S3)
2. Удаление старого плагина (То же, что и шаг 2 для S3)
3. Установка нового плагина (То же, что и шаг 3 для S3)

4. Миграция данных:

- Выполните ssh-подключение к узлу Pod старого плагина, скопируйте данные из старого каталога в новый со всеми правами файлов:

```
sudo cp -a /cpaas/internal-docker-registry/* /cpaas/image-registry/
```

5. Очистка старого плагина из [Marketplace/Cluster Plugins] (То же, что и шаг 4 для S3)

Source to Image

Обзор

Введение

Концепция Source to Image

Основные возможности

Основные преимущества

Сценарии применения

Ограничения использования

Архитектура

Политика жизненного цикла

Примечани

Примечания к

Установка

Установка сборок Alauda Container Platform

Предварительные требования

Процедура

Обновление

Обновление сборок Alauda Container Platform

Предварительные требования

Процедура

Руководства

Управление приложениями, созданными из кода

Основные возможности

Преимущества

Предварительные требования

Процедура

Связанные операции

Как сделать

Создание приложения из кода

Предварительные требования

Процедура

Обзор

Введение

Alauda Container Platform Builds — это облачный контейнерный инструмент, предоставляемый Alauda Container Platform, который объединяет возможности Source to Image (S2I) с автоматизированными пайплайнами. Он ускоряет переход предприятий к облачным нативным решениям, обеспечивая полностью автоматизированные CI/CD пайплайны с поддержкой нескольких языков программирования, включая Java, Go, Python и Node.js. Кроме того, Alauda Container Platform Builds предлагает визуальное управление релизами и бесшовную интеграцию с Kubernetes-native инструментами, такими как Helm и GitOps, обеспечивая эффективное управление жизненным циклом приложений от разработки до продакшена.

Содержание

Концепция Source to Image

Source to Image (S2I) — это инструмент и рабочий процесс для создания воспроизводимых контейнерных образов из исходного кода. Он внедряет исходный код приложения в заранее определённый builder-образ и автоматически выполняет такие шаги, как компиляция и упаковка, в итоге генерируя запускаемый контейнерный образ.

Это позволяет разработчикам больше сосредоточиться на разработке бизнес-логики, не беспокоясь о деталях контейнеризации.

Основные возможности

Alauda Container Platform Builds обеспечивает полный стек облачного нативного рабочего процесса от кода до приложения, поддерживая сборку на нескольких языках и визуальное управление релизами. Он использует возможности Kubernetes-native для преобразования исходного кода в запускаемые контейнерные образы, обеспечивая бесшовную интеграцию в комплексную облачную платформу.

- **Сборка на нескольких языках**: поддержка сборки приложений на различных языках программирования, таких как Java, Go, Python и Node.js, что удовлетворяет разнообразные потребности разработки.
- **Визуальный интерфейс**: предоставляет интуитивно понятный интерфейс, позволяющий легко создавать, настраивать и управлять задачами сборки без глубоких технических знаний.
- **Полное управление жизненным циклом**: охватывает весь жизненный цикл от коммита кода до развертывания приложения, автоматизируя сборку, деплой и операционное управление.
- **Глубокая интеграция**: бесшовно интегрируется с вашим продуктом Container Platform, обеспечивая единый опыт разработки.
- **Высокая расширяемость**: поддерживает пользовательские плагины и расширения для удовлетворения ваших специфических требований.

Основные преимущества

- **Ускоренная разработка**: оптимизирует процесс сборки, ускоряя доставку приложений.
- **Повышенная гибкость**: поддержка сборки на нескольких языках программирования.
- **Повышенная эффективность**: автоматизация процессов сборки и развертывания, снижая ручное вмешательство.

- **Повышенная надежность**: предоставляет подробные логи сборки и визуальный мониторинг для облегчения устранения неполадок.

Сценарии применения

Основные сценарии применения S2I включают:

- Веб-приложения
  S2I поддерживает различные языки программирования, такие как Java, Go, Python и Node.js. Используя возможности управления приложениями Alauda Container Platform, можно быстро создавать и развертывать веб-приложения, просто указав URL репозитория кода.
- CI/CD
  S2I бесшовно интегрируется с DevOps пайплайнами, используя Kubernetes-native инструменты, такие как Helm и GitOps, для автоматизации процессов сборки и развертывания образов. Это обеспечивает непрерывную интеграцию и непрерывное развертывание приложений.

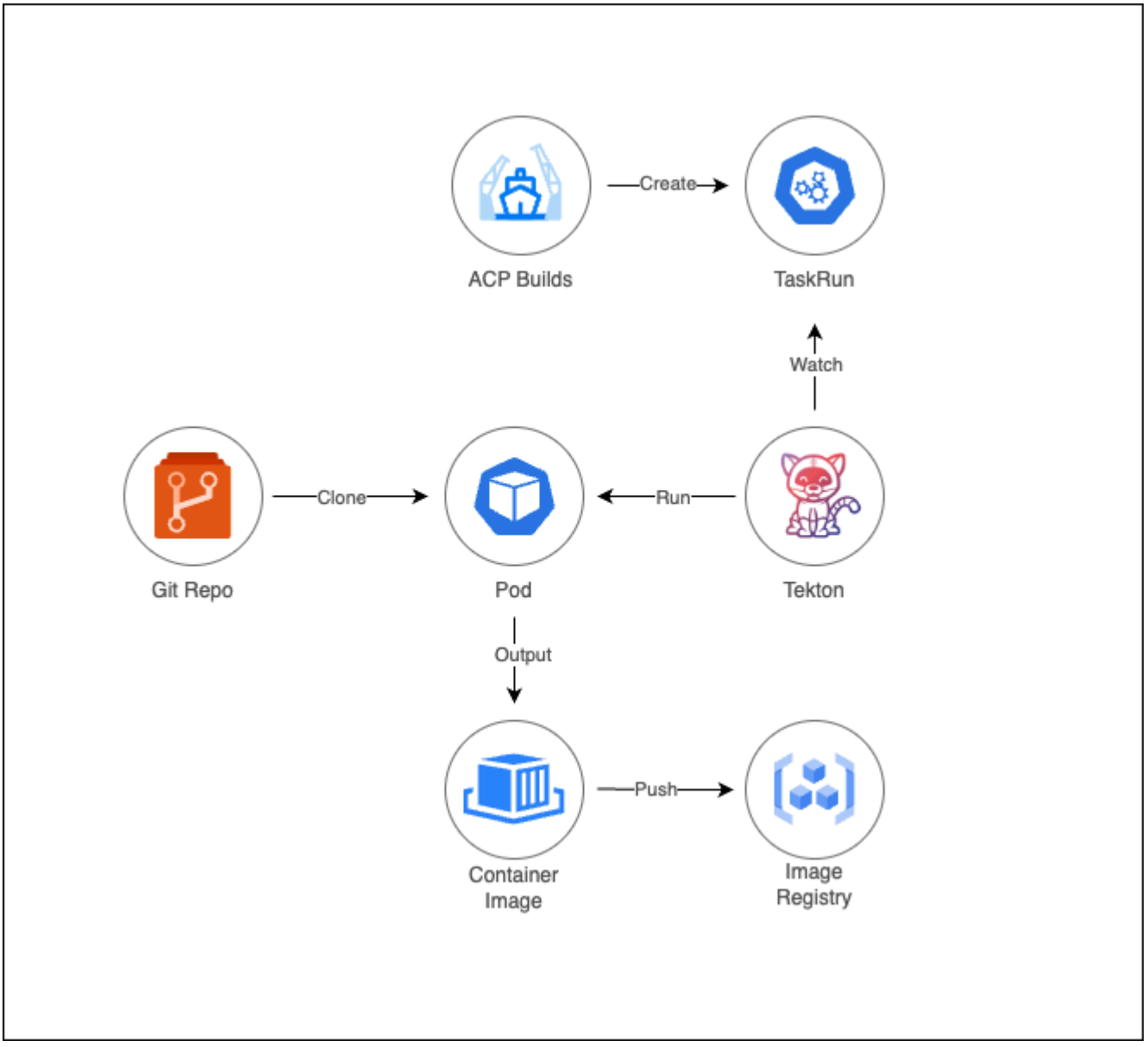
Ограничения использования

> Текущая версия поддерживает только языки Java, Go, Python и Node.js.

WARNING

Требования: Alauda DevOps Pipelines operator ↗ теперь доступен в кластере OperatorHub.

Архитектура



Возможность Source to Image (S2I) реализована через оператор **Alauda Container Platform Builds**, обеспечивающий автоматизированную сборку контейнерных образов из исходного кода репозитория Git и последующую отправку в указанный реестр образов. Основные компоненты включают:

- Оператор **Alauda Container Platform Builds**: Управляет полным жизненным циклом сборки и оркестрирует конвейеры Tekton.

- Конвейеры **Tekton**: Выполняют рабочие процессы S2I с помощью Kubernetes-нативных ресурсов `TaskRun`.

Примечания к выпуску

Содержание

Примечания к выпуску Alauda Container Platform Builds

Примечания к выпуску оператора **Alauda Container Platform Builds** описывают новые функции и улучшения, устаревшие функции и известные проблемы.

INFO

Оператор **Alauda Container Platform Builds** предоставляется как устанавливаемый компонент с отдельным циклом выпуска, отличным от основного Alauda Container Platform. Политика жизненного цикла оператора Alauda Container Platform Builds описывает совместимость версий.

Поддерживаемые версии

Версия	Версия Alauda Container Platform	Версия Alauda DevOps Pipelines
v1.1.1	**v4.2**,**v4.1**,**v4.0**	**v4.2**,**v4.1**
v1.1.0	**v4.1**,**v4.0**	**v4.1**

Примечания к выпуску v1.1

v1.1.1

1. Исправление уязвимостей безопасности.

v1.1.0

1. Исправление уязвимостей безопасности.
2. Независимый выпуск.

Политика жизненного цикла

График жизненного цикла версий

Ниже приведён график жизненного цикла выпущенных версий Alauda Container Platform Builds Operator:

Version	Release Date	End of Life
v1.1.1	2026-01-06	2028-01-06
v1.1.0	2025-08-15	2027-08-15

Установка

Установка сборок Alauda Container Platform

Установка сборок Alauda Container Platform

Содержание

Предварительные требования

> Alauda Container Platform Builds — это контейнерный инструмент, предлагаемый Alauda Container Platform, который объединяет сборку (с поддержкой Source to Image) и создание приложений.

1. Скачайте последнюю версию пакета **Alauda Container Platform Builds**, соответствующую вашей платформе. Если оператор **Alauda DevOps Pipelines** ещё не установлен в Kubernetes кластере, рекомендуется скачать его вместе с ним.
2. Используйте CLI-инструмент `violet` для загрузки пакетов **Alauda Container Platform Builds** и **Alauda DevOps Pipelines** в целевой кластер. Подробные инструкции по использованию `violet` доступны в разделе CLI.

Процедура

Установка оператора Alauda Container Platform Builds

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace** > **OperatorHub**.
3. Найдите оператора **Alauda Container Platform Builds**, нажмите **Install** и перейдите на страницу **Install**.

Параметры конфигурации:

Параметр	Рекомендуемая конфигурация
Channel	`Alpha`: По умолчанию выбран канал **alpha**.
Version	Выберите последнюю версию.
Installation Mode	`Cluster`: Один оператор используется для всех namespace в кластере для создания и управления инстансами, что снижает потребление ресурсов.
Namespace	`Recommended`: Рекомендуется использовать namespace **shipyard-operator**; он будет создан автоматически, если отсутствует.
Upgrade Strategy	Выберите `Manual`. • `Manual`: При появлении новой версии в OperatorHub • действие **Upgrade** не будет выполняться автоматически.

4. На странице **Install** выберите конфигурацию по умолчанию, нажмите **Install** и завершите установку оператора **Alauda Container Platform Builds**.

Установка инстанса Shipyard

1. Нажмите **Marketplace** > **OperatorHub**.

2. Найдите установленного оператора **Alauda Container Platform Builds**, перейдите в раздел **All Instances**.
3. Нажмите кнопку **Create Instance**, затем выберите карточку **Shipyard** в области ресурсов.
4. На странице настройки параметров инстанса можно использовать конфигурацию по умолчанию, если нет специальных требований.
5. Нажмите **Create**.

Проверка

- После успешного создания инстанса подождите примерно 20 минут, затем перейдите в **Container Platform** > **Applications** > **Applications** и нажмите **Create**.
- Вы должны увидеть пункт **Create from Code**. Это означает, что установка Alauda Container Platform Builds прошла успешно, и вы можете начать работу с S2I, следуя руководству Creating an application from Code.

Обновление

Обновление сборок Alauda Container Platform

Обновление сборок Alauda Container Platform

Содержание

Предварительные требования

> Alauda Container Platform Builds — это инструмент контейнеризации, предлагаемый Alauda Container Platform, который объединяет сборку (включая возможность Source to Image) и создание приложений.

1. Скачайте пакет новой версии **Alauda Container Platform Builds**, соответствующий вашей платформе.
2. Используйте CLI-инструмент `violet` для загрузки пакетов **Alauda Container Platform Builds** и **Alauda DevOps Pipelines** в целевой кластер. Подробные инструкции по использованию `violet` см. в разделе CLI.

Процедура

Обновление оператора Alauda Container Platform Builds

> **INFO**
>
> Если вы обновляетесь с версии v4.0 и ранее, сначала выполните миграцию **Alauda DevOps Tekton v3** на **Alauda DevOps Pipelines**. Подробности см. в руководстве по миграции ↗.

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace** > **OperatorHub**.
3. В навигационной панели выберите кластер, в котором установлен оператор.
4. Найдите оператора **Alauda Container Platform Builds** и откройте его страницу **Details**.
5. Нажмите **Confirm**, чтобы начать обновление, и дождитесь завершения процесса обновления оператора.

Руководства

Управление приложениями, созданными из кода

Управление приложениями, созданными из кода

Содержание

Основные возможности

- Введите URL репозитория кода для запуска процесса S2I, который преобразует исходный код в образ и публикует его как приложение.
- При обновлении исходного кода инициируйте действие **Rebuild** через визуальный интерфейс, чтобы обновить версию приложения одним кликом.

Преимущества

- Упрощает процесс создания и обновления приложений из кода.

- Снижает порог для разработчиков, устраняя необходимость разбираться в деталях контейнеризации.
- Обеспечивает визуальный процесс построения и управления эксплуатацией, облегчая локализацию, анализ и устранение проблем.

Предварительные требования

- Завершена установка Installing Alauda Container Platform Builds.
- Требуется доступ к репозиторию образов; если доступа нет, обратитесь к Администратору для Installing Alauda Container Platform Registry.

Процедура

1. В **Container Platform** перейдите в **Application** > **Application**.
2. Нажмите **Create**.
3. Выберите **Create from Code**.
4. Ознакомьтесь с описанием параметров ниже и заполните конфигурацию.

Раздел	Параметр	Описание
Репозиторий кода	**Тип**	• **Platform Integrated**: Выберите репозиторий кода, интегрированный с платформой и уже выделенный для текущего проекта; платформа поддерживает GitLab, GitHub и Bitbucket. • **Input**: Используйте URL репозитория кода, не интегрированного с платформой.

Название интегрированного проекта	Название проекта интеграционного инструмента, назначенного или связанного с текущим проектом Администратором.
Адрес репозитория	Выберите или введите адрес репозитория кода, в котором хранится исходный код.
Идентификатор версии	Поддерживается создание приложений на основе веток, тегов или коммитов в репозитории кода. В частности: • Если идентификатор версии — ветка, поддерживается создание приложений только по последнему коммиту в выбранной ветке. • Если идентификатор версии — тег или коммит, по умолчанию выбирается последний тег или коммит в репозитории. Однако при необходимости можно выбрать и другие версии.
Context dir	Необязательный каталог исходного кода, используемый как контекстный каталог для сборки.
Secret	При использовании входного репозитория кода можно при необходимости добавить секрет аутентификации.
Builder Image	• Образ, включающий конкретные среды выполнения языков программирования, библиотеки зависимостей и скрипты S2I. Основная задача — преобразование исходного кода в исполняемые образы приложений.

		• Поддерживаемые builder images включают: Golang, Java, Node.js и Python.
	Версия	Выберите версию среды выполнения, совместимую с вашим исходным кодом, чтобы обеспечить корректное выполнение приложения.
Build	**Тип сборки**	В настоящее время поддерживается только метод **Build** для создания образов приложений. Этот метод упрощает и автоматизирует сложный процесс сборки образов, позволяя разработчикам сосредоточиться исключительно на разработке кода. Общий процесс следующий: 1. После установки Alauda Container Platform Builds и создания экземпляра Shipyard система автоматически генерирует ресурсы на уровне кластера, такие как ClusterBuildStrategy, и определяет стандартизированный процесс сборки. Этот процесс включает подробные шаги сборки и необходимые параметры, что позволяет выполнять сборки Source-to-Image (S2I). Подробнее см. в: Installing Alauda Container Platform Builds 2. Создайте ресурсы типа Build на основе вышеуказанных стратегий и информации, предоставленной в форме. Эти ресурсы определяют стратегии сборки, параметры сборки, репозитории исходного кода, репозитории выходных

		образов и другую релевантную информацию. 3. Создайте ресурсы типа BuildRun для запуска конкретных экземпляров сборки, которые координируют весь процесс сборки. 4. После создания BuildRun система автоматически создаст соответствующий экземпляр ресурса TaskRun. Этот TaskRun запускает сборку конвейера Tekton и создает Pod для выполнения процесса сборки. Pod отвечает за фактическую работу сборки, которая включает: загрузку исходного кода из репозитория. Вызов указанного builder image. Выполнение процесса сборки.
	URL образа	После завершения сборки укажите адрес целевого репозитория образов для приложения.
Приложение	-	Заполните конфигурацию приложения по необходимости. Для подробностей обратитесь к описанию параметров в документации Creating applications from Image.
Сеть	-	• **Target Port**: Фактический порт, на котором приложение внутри контейнера слушает. При включенном внешнем доступе весь соответствующий трафик

		будет перенаправлен на этот порт для предоставления внешних сервисов. • **Другие параметры**: См. описание параметров в документации CreatingIngress.
Метки и аннотации	-	Заполните соответствующие метки и аннотации по необходимости.

5. После заполнения параметров нажмите **Create**.
6. Вы можете просмотреть соответствующее развертывание на странице **Details**.

Связанные операции

Build

После создания приложения соответствующую информацию можно просмотреть на странице деталей.

Параметр	Описание
Build	Нажмите на ссылку, чтобы просмотреть конкретную информацию о ресурсе сборки (Build) и задаче сборки (BuildRun) и их YAML.
Start Build	При сбое сборки или изменении исходного кода можно нажать эту кнопку для повторного запуска задачи сборки.

Как сделать

Создание приложения из кода

Создание приложения из кода

> Использование мощных возможностей установки Alauda Container Platform Builds для реализации полного процесса от исходного кода Java до создания приложения, а в конечном итоге — для эффективного запуска приложения в контейнеризованном виде на Kubernetes.

Содержание

Предварительные требования

Перед использованием данной функции убедитесь, что:

- Установлен Alauda Container Platform Builds
- На платформе доступен репозиторий образов. Если нет, обратитесь к Администратору для установки ACP Registry

Процедура

1. В **Container Platform** нажмите **Applications** > **Applications**.
2. Нажмите **Create**.

3. Выберите **Create from Code**.

4. Заполните конфигурацию согласно параметрам ниже:

Параметр	Рекомендуемая конфигурация
Code Repository	**Тип**: `Input` **Repository URL**: `https://github.com/alauda/spring-boot-hello-world`
Build Method	`Build`
Image Repository	Обратитесь к Администратору.
Application	**Application**: `spring-boot-hello-world` **Name**: `spring-boot-hello-world` **Resource Limits**: Используйте значение по умолчанию.
Network	**Target Port**: `8080`

5. После заполнения параметров нажмите **Create**.

6. Вы можете проверить статус соответствующего приложения на странице **Details**.

Стратегия изоляции узлов

Стратегия изоляции узлов предоставляет стратегию изоляции узлов на уровне проекта, которая позволяет проектам использовать узлы кластера эксклюзивно.

Введение

Архитектура

Основные понятия

Руководства

Создание стратегии изоляции узлов

Создание стратегии изоляции узлов

Удаление стратегии изоляции узлов

Разрешения

Разрешения

Введение

Node Isolation Strategy предоставляет стратегию изоляции узлов на уровне проекта, которая позволяет проектам использовать узлы кластера эксклюзивно.

Содержание

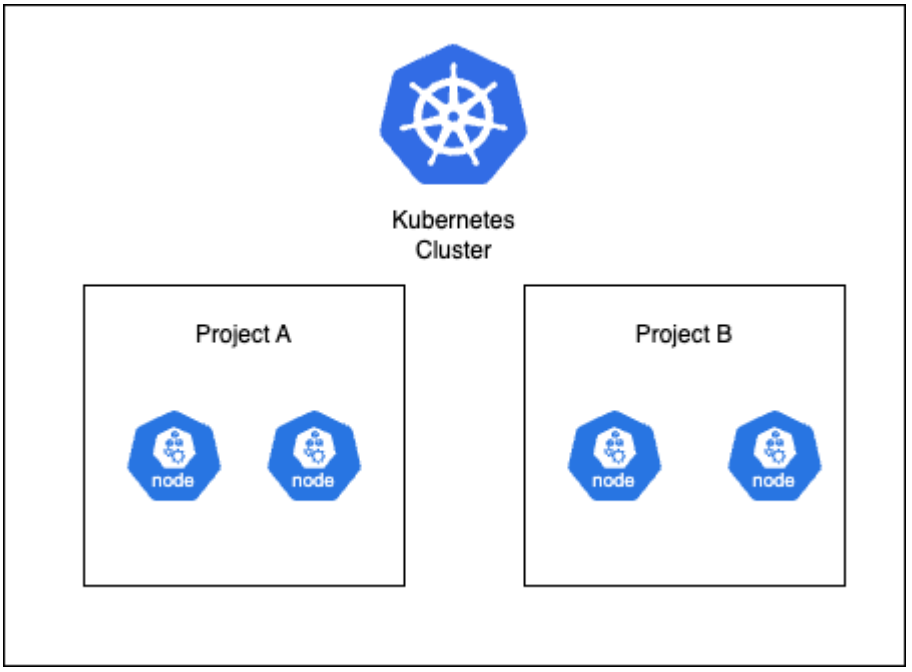
Преимущества

Удобное выделение узлов проектам в эксклюзивном или совместном режиме, предотвращая конкуренцию за ресурсы между проектами.

Сценарии применения

Node Isolation Strategy подходит для сценариев, где требуется усиленная изоляция ресурсов между проектами и необходимо предотвратить использование узлов компонентами других проектов, что может привести к ограничению ресурсов или невозможности выполнения требований к производительности.

Архитектура



Стратегия изоляции узлов реализована на основе компонента Container Platform Cluster Core и обеспечивает возможность изоляции узлов между проектами путём выделения узлов на каждом кластере рабочих нагрузок. Когда контейнеры создаются в проекте, они принудительно планируются на узлы, выделенные именно этому проекту.

Основные понятия

Основные понятия

Содержание

Изоляция узлов

Изоляция узлов означает изоляцию узлов в кластере для предотвращения одновременного использования контейнерами из разных проектов одного и того же узла, что позволяет избежать конфликтов ресурсов и ухудшения производительности.

Руководства

Создание стратегии изоляции узлов

Создание стратегии изоляции узлов

Создайте политику изоляции узлов для текущего кластера, позволяющую указанным проектам иметь эксклюзивный доступ к узлам сгруппированных ресурсов внутри кластера, тем самым ограничивая узлы, на которых могут запускаться Pod'ы в рамках проекта, достигая физической изоляции ресурсов между проектами.

Содержание

Создание стратегии изоляции узлов

1. В левой навигационной панели нажмите **Security** > **Node Isolation Strategy**.
2. Нажмите **Create Node Isolation Strategy**.
3. Следуйте инструкциям ниже для настройки соответствующих параметров.

Параметр	Описание
Project Exclusivity	Включение или отключение переключателя для узлов, содержащихся в политике изоляции проекта, настроенной в стратегии; нажмите для переключения в положение вкл или выкл, по умолчанию включено. Когда переключатель включен, только Pod'ы в указанном проекте из политики могут запускаться на узлах, включённых в политику; при

Параметр	Описание
	выключенном переключателе Pod'ы из других проектов текущего кластера также могут запускаться на узлах, включённых в политику, помимо указанного проекта.
Project	Проект, для которого настроено использование узлов в политике. Нажмите на выпадающий список **Project** и отметьте флажок перед названием проекта для выбора нескольких проектов. **Примечание**: Для проекта может быть установлена только одна политика изоляции узлов; если проект уже назначен политике изоляции узлов, его нельзя выбрать; Поддерживается ввод ключевых слов в выпадающем списке для фильтрации и выбора проектов.
Node	IP-адреса вычислительных узлов, выделенных для использования проектом в политике. Нажмите на выпадающий список **Node** и отметьте флажок перед названием узла для выбора нескольких узлов. **Примечание**: Узел может принадлежать только одной политике изоляции; если узел уже принадлежит другой политике изоляции, его нельзя выбрать; Поддерживается ввод ключевых слов в выпадающем списке для фильтрации и выбора узлов.

4. Нажмите **Create**.

   Примечание:

   - После создания политики существующие Pod'ы в проекте, не соответствующие текущей политике, будут запланированы на узлы, включённые в текущую политику, после их пересоздания;
   - При включённом параметре **Project Exclusivity** Pod'ы, уже запущенные на узлах, не будут автоматически выселены; при необходимости выселения требуется ручное планирование.

Удаление стратегии изоляции узлов

Примечание: После удаления политики изоляции узлов проект больше не будет ограничен запуском на определённых узлах, и узлы перестанут использоваться эксклюзивно этим проектом.

1. В левой навигационной панели нажмите **Security** > **Node Isolation Strategy**.
2. Найдите политику изоляции узлов, нажмите ⋮ > **Delete**.

Разрешения

Функция	Действие	Platform Administrator	Platform auditors	Project Manager	Namespace Administrator
nodegroups `acp-nodegroups`	Просмотр	✓	✓	✓	✓
	Создать	✓	×	×	×
	Обновить	✓	×	×	×
	Удалить	✓	×	×	×

Часто задаваемые вопросы

Содержание

Почему не должно быть нескольких ResourceQuota в одном namespace при его импорте?

При импорте namespace, если в нем содержится несколько ресурсов ResourceQuota, платформа выберет наименьшее значение для каждого элемента квоты среди всех ResourceQuota и объединит их, в итоге создав один ResourceQuota с именем `default`.

Пример:

Namespace `to-import`, который нужно импортировать, содержит следующие ресурсы `resourcequota`:

```
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: a
  namespace: to-import
spec:
  hard:
    requests.cpu: "1"
    requests.memory: "500Mi"
    limits.cpu: "3"
    limits.memory: "1Gi"
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: b
  namespace: to-import
spec:
  hard:
    requests.cpu: "2"
    requests.memory: "300Mi"
    limits.cpu: "2"
    limits.memory: "2Gi"
```

После импорта namespace `to-import` в нем будет создан следующий ResourceQuota с именем `default`:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: default
  namespace: to-import
spec:
  hard:
    requests.cpu: "1"
    requests.memory: "300Mi"
    limits.cpu: "2"
    limits.memory: "1Gi"
```

Для каждого ResourceQuota квоты ресурсов берутся как минимальное значение между `a` и `b`.

Когда в namespace существует несколько ResourceQuota, Kubernetes проверяет каждую ResourceQuota независимо. Поэтому после импорта namespace рекомендуется удалить все ResourceQuota, кроме `default`. Это помогает избежать сложностей в расчетах квот из-за нескольких ResourceQuota, что может легко привести к ошибкам.

Почему не должно быть нескольких LimitRange в одном namespace при его импорте?

При импорте namespace, если в нем содержится несколько ресурсов LimitRange, платформа не может объединить их в один LimitRange. Поскольку Kubernetes проверяет каждый LimitRange независимо, а поведение выбора значений по умолчанию из какого LimitRange будет использоваться Kubernetes непредсказуемо.

Если в namespace содержится только один LimitRange, платформа создаст LimitRange с именем `default` и значениями из этого LimitRange.

Поэтому перед импортом namespace в нем должен существовать только один LimitRange. После импорта namespace рекомендуется удалить все LimitRange, кроме того, что называется `default`, чтобы избежать непредсказуемого поведения из-за нескольких LimitRange.