Menu

Разработчик

Обзор

Быстрый старт

Создание приложений

Построение архитектуры приложения

Введение в построение приложения

Основные компоненты

Основные понятия

Пространства имён

Создание приложений

Эксплуатация и сопровождение приложений

Рабочие нагрузки

Работа с Helm charts

1. Понимание Helm

2 Развертывание Helm Charts как приложений через CLI

3. Развертывание Helm Charts как приложений через UI

Конфигурации

Наблюдаемость приложения

Как сделать

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

Руководство пользователя

Source to Image

Обзор

Установка

Обновление

Руководства

Как сделать

Стратегия изоляции узлов

Введение

Преимущества

Сценарии применения

Архитектура

Основные понятия

Руководства

Разрешения

Часто задаваемые вопросы

Часто задаваемые вопросы

Почему при импорте namespace не должно быть нескольких ResourceQuota?

Почему при импорте namespace не должно быть нескольких LimitRanges?

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 несколькими 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 определённой версии для быстрого восстановления.
- Удаление приложений: Удаление приложения одновременно удаляет само приложение и все его непосредственно 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 отдельные представления для приложений, Workloads и Pods, а также поддержка настройки панелей мониторинга под конкретные операционные требования.

Управление рабочими нагрузками Kubernetes

Поддержка основных типов рабочих нагрузок:

- Deployments: Управление безсостояночными приложениями с возможностью поэтапных обновлений.
- 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.

Menu

Быстрый старт

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 методологий.

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
- Требуемые права:
  - Создание проектов/пространств имён
  - Интеграция репозиториев 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	Установить границы изоляции ресурсов
2	Configure Image Repository	Настроить источники контейнерных образов
3	Create application via Deployment	Создать workload Deployment
4	Expose Service via NodePort	Настроить сервис NodePort
5	Validate Application Accessibility	Проверить доступность приложения

Procedure

Create namespace

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

Prerequisites

- Права 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 вновь созданные учётные данные

3. Назначьте реестр целевому проекту платформы

Method 2: External Registry Services

- Используйте общедоступные URL реестров (например, Docker Hub)
- Пример: `index.docker.io/library/nginx:latest`

Verification Requirement

- Сеть кластера должна иметь исходящий доступ к конечным точкам реестра

Create application via Deployment

> Deployment обеспечивает декларативное обновление реплик Pod.

Creation Process

1. В представлении **Container Platform**:

- Выберите целевое пространство имён через селектор

2. Перейдите в **Workloads** > **Deployments**
3. Нажмите **Create Deployment**
4. Укажите источник образа:

- Выберите интегрированный реестр *или*
- Введите внешний URL образа (например, `index.docker.io/library/nginx:latest`)

5. Настройте идентичность 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

Parameter	Значение
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 хост-порт для доступа.

1. Нажмите **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`
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), сопоставляемый 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

Menu

Создание приложений

Построение архитектуры приложения

Построение архитектуры приложения

Введение в построение приложения

Основные компоненты

Основные понятия

Типы приложений

Custom Applications

Понимание пользовательских приложений

Архитектурный дизайн CRD пользовательского приложения

Типы рабочих нагрузок

Понимание параметров

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Устранение распространённых проблем

Расширенные шаблоны использования

Понимание переменных окружения

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Расширенные шаблоны использования

Описание единиц ресурсов

Пространства имён

Создание пространств имён

Понимание пространств имён

Создание пространств имён с помощью веб-консоли

Создание пространства имён с помощью CLI

Импорт пространств имён

Overview

Use Cases

Prerequisites

Procedure

Resource Quota

Понимание Resource Requests и Limits

Квоты

Квоты ресурсов аппаратных ускорителей

Limit Range

Понимание Limit Range

Создание Limit Range с помощью CLI

Pod Security Admission

Режимы безопасности

Стандарты безопасности

Конфигурация

Назначение UID/GID

Включение назначения UID/GID

Проверка назначения UID/GID

Коэффициент Overcommit

Понимание коэффициента overcommit ресурсов в Namespace

Определение CRD

Создание коэффициента overcommit с помощью CLI

Создание/обновление коэффициента overcommit через веб-консоль

Управление участниками пространства имён

Импорт участников

Добавление участников

Удаление участников

Обновление Namespaces

Обновление Quotas

Обновление Container LimitRanges

Обновление Pod Security Admission

Удаление/Исключение Namespaces

Удаление Namespaces

Исключение Namespaces

Создание приложений

Создание приложений из образа

Предварительные требования

Процедура 1 — Workloads

Процедура 2 — Services

Процедура 3 — Ingress

Операции управления приложением

Справочная информация

Создание приложений из Chart

Меры предосторожности

Предварительные требования

Процедура

Справка по анализу статуса

Создание приложений из YAML

Меры предосторожности

Предварительные условия

Процедура

Создание приложений из кода

Требования

Процедура

Creating applications from Operator Backed

Понимание Operator Backed Application

Создание Operator Backed Application через веб-консоль

Устранение неполадок

Создание приложений с использованием CLI

Предварительные требования

Процедура

Пример

Справка

Эксплуатация и сопровождение приложений

Развертывание приложений

Описание статуса

Приложения

KEDA (Kubernetes Event-driven Autoscaling)

Настройка HPA

Понимание Horizontal Pod Autoscalers

Предварительные требования

Создание Horizontal Pod Autoscaler

Правила расчёта

Запуск и остановка приложений

Запуск приложения

Остановка приложения

Настройка VerticalPodAutoscaler (VPA)

Понимание VerticalPodAutoscalers

Предварительные требования

Создание VerticalPodAutoscaler

Последующие действия

Настройка CronHPA

Понимание Cron Horizontal Pod Autoscalers

Предварительные требования

Создание Cron Horizontal Pod Autoscaler

Объяснение правил расписания

Обновление приложений

Импорт ресурсов

Удаление/пакетное удаление ресурсов

Экспорт приложений

Экспорт Helm Charts

Экспорт YAML локально

Экспорт YAML в репозиторий кода (Alpha)

Обновление и удаление Chart-приложений

Важные замечания

Предварительные требования

Описание анализа состояния

Управление версиями приложений

Создание снимка версии

Откат к исторической версии

Удаление приложений

Проверки состояния

Понимание проверок состояния

Пример YAML файла

Параметры конфигурации проверок состояния через веб-консоль

Устранение неполадок с провалами проб

Рабочие нагрузки

Deployments

Понимание Deployments

Создание Deployments

Управление Deployments

Устранение неполадок с помощью CLI

DaemonSets

Понимание DaemonSets

Создание DaemonSets

Управление DaemonSets

StatefulSets

Понимание StatefulSets

Создание StatefulSets

Управление StatefulSets

CronJobs

Понимание CronJobs

Создание CronJobs

Немедленное выполнение

Удаление CronJobs

Jobs

Понимание Jobs

Пример YAML файла

Обзор выполнения

Pods

Понимание Pod'ов

Пример YAML файла

Управление Pod с помощью CLI

Управление Pod с помощью веб-консоли

Контейнеры

Понимание контейнеров

Понимание эфемерных контейнеров

Взаимодействие с контейнерами

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 через CLI

Способы использования ConfigMap в Pod

ConfigMap и Secret

Настройка Secrets

Понимание Secrets

Создание Secret типа Opaque

Создание Secret типа Docker registry

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 приложения

Мониторинговые панели

Предварительные требования

Панели мониторинга на уровне Namespace

Мониторинг на уровне рабочих нагрузок

Логи

Процедура

События

Процедура

Интерпретация записей событий

Как сделать

Настройка правил срабатывания планировщика задач

Преобразование времени

Запись выражений Crontab

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 в `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 универсальный контроллер внутри `business clusters`, обеспечивая критически важные операции на уровне кластера:

- **Управление webhook**: Реализует admission webhook для валидации ресурсов, включая применение правил `resources ratio` и политики `resource labeling` и др.
- **Синхронизация статусов**: Поддерживает согласованность между распределёнными компонентами через:
  - согласование статуса применения `Helm chart`
  - синхронизацию `Project quota`
  - обновление статусов `Application` (запись в поля application.status)

Captain controller manager

Выступает в роли контроллера управления жизненным циклом приложений `Helm chart`, работающего исключительно в `global cluster`. Его обязанности включают:

- **Установку chart**: 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 аккаунты арендаторов через изоляцию пространств имён
  - Управляет правами доступа к ресурсам для каждого арендатора
  - Обеспечивает изоляцию представлений для каждого арендатора
- Работает исключительно в `global cluster`, выступая в качестве единой точки управления мультикластерными операциями

Menu

Основные понятия

Типы приложений

Custom Applications

Понимание пользовательских приложений

Архитектурный дизайн CRD пользовательского приложения

Типы рабочих нагрузок

Понимание параметров

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Устранение распространённых проблем

Расширенные шаблоны использования

Понимание переменных окружения

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Расширенные шаблоны использования

Описание единиц ресурсов

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 с помощью 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 на основе 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 преимущества автоматического развертывания, масштабирования, обновлений и обслуживания, 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 их 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, позволяя выполнять атомарные операции.

  1. **Абстракция на уровне приложения и агрегирование состояния**

- Скрывает детали низкоуровневых ресурсов (например, статус реплик Pod). Разработчики могут напрямую через ресурс Application отслеживать общее состояние приложения (например, соотношение готовых эндпоинтов, согласованность версий).

- Поддерживает декларации зависимостей между компонентами (например, сервис базы данных должен запускаться раньше сервиса приложения) для обеспечения порядка и координации инициализации ресурсов.

  1. **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).

  1. **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.
Стандартизация	Унифицирует стандарты описания приложений через Application CRD, устраняя хаос управления, вызванный фрагментацией YAML.

Измерение	Ценностное предложение
Совместимость экосистемы	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 приложениями:

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 связана с Application CRD для полной трассировки изменений.

Определение Application CRD

Application CRD использует поле `spec.componentKinds` для объявления типов ресурсов Kubernetes (например, Deployment, Service), что позволяет управлять 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 defines the versioned schema of this representation
            of an object. Servers should convert recognized schemas to the latest
            internal value, and may reject unrecognized values. More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#resources'
          type: string
        kind:
          description: 'Kind is a string value representing the REST resource this
            object represents. Servers may infer this from the endpoint the client
            submits requests to. Cannot be updated. In CamelCase. More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#types-kinds'
          type: string
        metadata:
          description: 'Metadata is a object value representing the metadata of the
kubernetes resource.
            More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#metadata'
          type: object
        spec:
          properties:
            assemblyPhase:
              description: |
                The installer can set this field to indicate that the application's
```

```
components
                are still being deployed ("Pending") or all are deployed already
("Succeeded"). When the
                application cannot be successfully assembled, the installer can set this
field to "Failed".'
              type: string
            componentKinds:
              description: |
                This array of GroupKinds is used to indicate the types of resources that
the
                application is composed of. As an example an Application that has a
service and a deployment
                would set this field to [{"group":"core","kind": "Service"},
{"group":"apps","kind":"Deployment"}]
              items:
                description: 'The item of the GroupKinds, with a structure like \"
{"group":"core","kind": "Service"}\"'
                type: object
              type: array
            descriptor:
              properties:
                description:
                  description: 'A short, human readable textual description of the
Application.'
                  type: string
                icons:
                  description: 'A list of icons for an application. Icon information
includes the source, size, and mime type.'
                  items:
                    properties:
                      size:
                        description: 'The size of the icon.'
                        type: string
                      src:
                        description: 'The source of the icon.'
                        type: string
                      type:
                        description: 'The mime type of the icon.'
                        type: string
                    required:
                    - src
                    type: object
                  type: array
                keywords:
```

```
                  description: 'A list of keywords that identify the application.'
                  items:
                    type: string
                  type: array
                links:
                  description: 'Links are a list of descriptive URLs intended to be used
to surface additional documentation, dashboards, etc.'
                  items:
                    properties:
                      description:
                        description: 'The description of the link.'
                        type: string
                      url:
                        description: 'The url of the link.'
                        type: string
                    type: object
                  type: array
                maintainers:
                  description: 'A list of the maintainers of the Application. Each
maintainer has a
                    name, email, and URL. This field is meant for the distributors of the
Application
                    to indicate their identity and contact information.'
                  items:
                    properties:
                      email:
                        description: 'The email of the maintainer.'
                        type: string
                      name:
                        description: 'The name of the maintainer.'
                        type: string
                      url:
                        description: 'The url to contact the maintainer.'
                        type: string
                    type: object
                  type: array
                notes:
                  description: 'Notes contain human readable snippets intended as a quick
start
                    for the users of the Application. They may be plain text or
CommonMark markdown.'
                  type: string
                owners:
                  items:
```

```
                  properties:
                    email:
                      description: 'The email of the owner.'
                      type: string
                    name:
                      description: 'The name of the owner.'
                      type: string
                    url:
                      description: 'The url to contact the owner.'
                      type: string
                  type: object
                type: array
              type:
                description: 'The type of the application (e.g. WordPress, MySQL,
Cassandra).
                  You can have many applications of different names in the same
namespace.
                  They type field is used to indicate that they are all the same type
of application.'
                type: string
              version:
                description: 'A version indicator for the application (e.g. 5.7 for
MySQL version 5.7).'
                type: string
            type: object
          info:
            description: 'Info contains human readable key-value pairs for the
Application.'
            items:
              properties:
                name:
                  description: 'The name of the information.'
                  type: string
                type:
                  description: 'The type of the information.'
                  type: string
                value:
                  description: 'The value of the information.'
                  type: string
                valueFrom:
                  description: 'The value reference from other resource.'
                  properties:
                    configMapKeyRef:
                      description: 'The config map key reference.'
```

```
                    properties:
                      key:
                        type: string
                    type: object
                  ingressRef:
                    description: 'The ingress reference.'
                    properties:
                      host:
                        description: 'The host of the ingress reference.'
                        type: string
                      path:
                        description: 'The path of the ingress reference.'
                        type: string
                    type: object
                  secretKeyRef:
                    description: 'The secret key reference.'
                    properties:
                      key:
                        type: string
                    type: object
                  serviceRef:
                    description: 'The service reference.'
                    properties:
                      path:
                        description: 'The path of the service reference.'
                        type: string
                      port:
                        description: 'The port of the service reference.'
                        format: int32
                        type: integer
                    type: object
                  type:
                    type: string
                type: object
              type: object
            type: array
          selector:
            description: 'The selector is used to match resources that belong to the
Application.
              All of the applications resources should have labels such that they match
this selector.
              Users should use the app.kubernetes.io/name label on all components of
the Application
              and set the selector to match this label. For instance,
```

```
                {"matchLabels": [{"app.kubernetes.io/name": "my-cool-app"}]} should be
used as the selector
                for an Application named "my-cool-app", and each component should contain
a label that matches.'
              type: object
          type: object
        status:
          description: 'The status summarizes the current state of the object.'
          properties:
            observedGeneration:
              description: 'The observedGeneration is the generation most recently
observed by the component
                responsible for acting upon changes to the desired state of the
resource.'
              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 сквозной аудиторской трассировки.

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: applicationhistories.app.k8s.io
spec:
  group: app.k8s.io
  names:
    kind: ApplicationHistory
    listKind: ApplicationHistoryList
    plural: applicationhistories
    singular: applicationhistory
  scope: Namespaced
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description: 'APIVersion defines the versioned schema of this representation
            of an object. Servers should convert recognized schemas to the latest
            internal value, and may reject unrecognized values. More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#resources'
          type: string
        kind:
          description: 'Kind is a string value representing the REST resource this
            object represents. Servers may infer this from the endpoint the client
            submits requests to. Cannot be updated. In CamelCase. More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#types-kinds'
          type: string
        metadata:
          description: 'Metadata is a object value representing the metadata of the
kubernetes resource.
            More info:
https://github.com/kubernetes/community/blob/master/contributors/devel/sig-
architecture/api-conventions.md#metadata'
          type: object
        spec:
          properties:
            changeCause:
              description: 'The change cause of the application to generate the
ApplicationHistory.'
              type: string
            creationTimestamp:
```

```
              description: 'The creation timestamp of the application history.'
              format: date-time
              type: string
            resourceDiffs:
              description: 'The resource differences between the current and last version
of application. It contains 3 types of diff: `create`,
                `delete` and `update`. The item of the diff compose of the kind and name
of the diff resource object.'
              properties:
                create:
                  items:
                    properties:
                      kind:
                        description: 'The kind of the created resource.'
                        type: string
                      name:
                        description: 'The name of the created resource.'
                        type: string
                    type: object
                  type: array
                delete:
                  items:
                    properties:
                      kind:
                        description: 'The kind of the deleted resource.'
                        type: string
                      name:
                        description: 'The name of the deleted resource.'
                        type: string
                    type: object
                  type: array
                update:
                  items:
                    properties:
                      kind:
                        description: 'The kind of the updated resource.'
                        type: string
                      name:
                        description: 'The name of the updated resource.'
                        type: string
                    type: object
                  type: array
              type: object
            revision:
```

```
              description: |
                The revision number of the application history. It's an integer that will
be incremented on
                every change of the application.'
              type: integer
            user:
              description: 'The user name who triggered the change of the application.'
              type: string
            yaml:
              description: |
                The YAML string of the snapshot of the application and it's components.
              type: string
          type: object
        status:
          description: 'The status summarizes the current state of the object.'
          properties:
            observedGeneration:
              description: 'The observedGeneration is the generation most recently
observed by the component
                responsible for acting upon changes to the desired state of the
resource.'
              format: int64
              type: integer
          type: object
      type: object
  version: v1beta1
  versions:
  - name: v1beta1
    served: true
    storage: true
```

Menu

Типы рабочих нагрузок

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

- Deployment: Наиболее часто используемый контроллер рабочих нагрузок 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 рабочих нагрузок 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 на основе времени с использованием cron-выражений. Система автоматически создаёт Jobs по расписанию, идеально подходит для периодических задач, таких как резервное копирование, генерация отчётов и очистка.
- Job: Рабочая 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 рабочих нагрузок через веб-консоль, Kubernetes также поддерживает прямое управление низкоуровневыми ресурсами 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 Docker-образах
- 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 Docker

В терминологии Docker:

- **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 в 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 off;" -c
"/custom/nginx.conf"

# Несколько параметров
kubectl run myapp --image=myapp:latest --restart=Never -- --port=8080 --log-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 yaml >
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"
```

Лучшие практики

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 параметров и документацию приложения
```

2. Переопределение параметров 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/sh

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

Расширенные шаблоны использования

1. Условные параметры с init-контейнерами

```yaml
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. Понимание правильного использования параметров позволяет создавать гибкие, переиспользуемые и поддерживаемые развёртывания, адаптирующиеся к различным 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 приложения.

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 образа
- стандартный способ передачи конфигурационных данных в приложения
- 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 в приложении** (наименьший приоритет)

Сценарии использования

1. Конфигурация приложения

Базовые настройки приложения:

```yaml
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 сред

Различные конфигурации для разных сред:

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

---
# Производственная 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 конкретной среды
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-config-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 обнаружением сервисов

```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
```

Pod с несколькими контейнерами и 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

```yaml
# ✅ 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 окружения
  value: "info"
- name: FEATURE_FLAG_NEW_UI  # Префикс для связанных переменных
  value: "true"

# ❌ Избегайте неясных или непоследовательных имён
# - name: db              # Слишком короткое
# - name: databaseHost    # Несогласованный стиль написания
# - 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 умолчанию
    - 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

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 тем, какой процесс 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 Docker-образах
- Определяют главный процесс (PID 1) внутри контейнера
- Работают совместно с параметрами (args) для формирования полной командной строки

Взаимосвязь с Docker и параметрами

Понимание взаимосвязи между инструкциями Docker и полями Kubernetes:

Docker	Kubernetes	Назначение
ENTRYPOINT	`command`	Определяет исполняемый файл
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	Docker Image	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 команды по умолчанию для запуска оболочки для 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 3600"

# 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 и сбоях в пайпах

      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: $(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: $(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 (millicore). 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. Виртуальная видеопамять GPU; 1 единица видеопамяти равна 256 Mi. Поддерживаются положительные целые числа.

Menu

Пространства имён

Создание пространств имён

Понимание пространств имён

Создание пространств имён с помощью веб-консоли

Создание пространства имён с помощью CLI

Импорт пространств имён

Overview

Use Cases

Prerequisites

Procedure

Resource Quota

Понимание Resource Requests и Limits

Квоты

Квоты ресурсов аппаратных ускорителей

Limit Range

Понимание Limit Range

Создание Limit Range с помощью CLI

Pod Security Admission

Режимы безопасности

Стандарты безопасности

Конфигурация

Назначение UID/GID

Включение назначения UID/GID

Проверка назначения UID/GID

Коэффициент Overcommit

Понимание коэффициента overcommit ресурсов в Namespace

Определение CRD

Создание коэффициента overcommit с помощью CLI

Создание/обновление коэффициента overcommit через веб-консоль

Управление участниками пространства имён

Импорт участников

Добавление участников

Удаление участников

Обновление Namespaces

Обновление Quotas

Обновление Container LimitRanges

Обновление Pod Security Admission

Удаление/Исключение Namespaces

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 внутри пространства имён, но не обязательно уникальными между разными пространствами имён. Область действия на основе пространства 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 с помощью веб-консоли

> 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, будет расходоваться квота, установленная здесь.

   ВНИМАНИЕ:

   - Квота ресурсов пространства имён наследуется от выделенной проекту квоты 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 можно также задавать квоты по памяти.

Определения единиц GPU:

- **vGPU Units**: 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 данного типа, выделенные проекту**, без явных ограничений.

Описание параметров квоты

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

Категория	Тип квоты	Значение и единица	Описание
			связанных с выбранным 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 смотрите в разделе Extended Resources Quotas description.	-	Эта категория не отображается, если отсутствует соответствующий конфигурационный словарь.
Другие квоты	Введите пользовательские квоты; правила ввода смотрите 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

Категория	Тип квоты	Значение и единица	Описание
			• requests.memory • pods • cpu • memory

6. (Опционально) Настройте **Container Limit Range**; подробности смотрите в разделе Limit Range.

7. (Опционально) Настройте **Pod Security Admission**; подробности смотрите 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 позволяют фильтровать и сортировать пространства имён.

9. Нажмите **Create**.

Создание пространства имён с помощью CLI

Примеры YAML-файлов

example-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: example
  labels:
    pod-security.kubernetes.io/audit: baseline # Option, to ensure security, it is recommended to choose the baseline or restricted mode.
    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

```
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-файлы

```
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 --
default='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 наследуемые политики на уровне проекта (например, Resource Quotas), единый мониторинг и централизованный контроль управления.

Важные замечания:

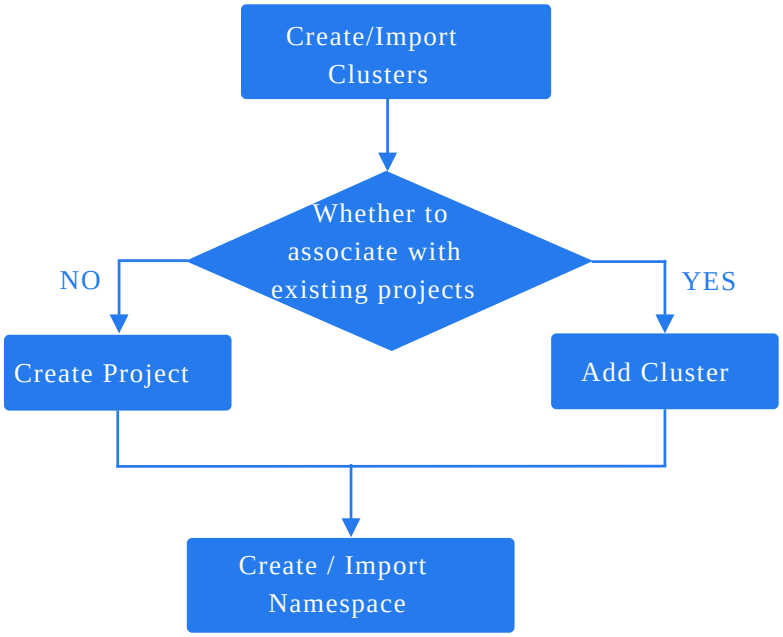
- Namespace может быть связан только с одним проектом в любой момент времени.

- 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 вы можете использовать функцию **Import Namespace** для ассоциации 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**, отсоединённый от одного **проекта**, может быть без проблем повторно ассоциирован 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 обеспечения управления на уровне платформы, включая видимость и централизованное управление.

Prerequisites

- Namespace в данный момент не управляется ни одним из существующих проектов 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 разделе **Project Management** выберите ***название проекта***, в который необходимо импортировать namespace.
2. Перейдите в **Namespaces** > **Namespaces**.
3. Нажмите 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, что 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 помечен 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 пользовательских имен квот должен соответствовать следующим требованиям:

- Если имя пользовательской квоты не содержит слеша (/): 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 слеш (/): имя делится на две части: префикс и имя, в форме: 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 **ограничение ресурсов на уровне контейнера или Pod**. Он устанавливает значения запросов по умолчанию, лимитов и максимальных значений для контейнеров или Pod, созданных после создания или обновления LimitRange, при этом постоянно отслеживает использование ресурсов контейнерами, чтобы гарантировать, что никакие ресурсы не превышают определённые максимальные значения в пространстве имён.

> Запрос ресурсов контейнера — это соотношение между лимитами ресурсов и overcommitment кластера. Значения запросов ресурсов служат ориентиром и критерием для scheduler при планировании контейнеров. Scheduler проверяет доступные ресурсы для каждого узла (общие ресурсы минус сумма запросов ресурсов контейнеров в Pod, запланированных на узле). Если суммарные запросы ресурсов нового контейнера Pod превышают оставшиеся доступные ресурсы узла, этот Pod не будет запланирован на данном узле.

LimitRange — это 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 максимальные значения ресурсов и соотношения, определённые в любом LimitRange, присутствующем в пространстве имён.

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 --
default='cpu=100m,memory=100Mi' --default-request='cpu=50m,memory=50Mi' --
max='cpu=1000m,memory=1000Mi'
```

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.

Содержание

Режимы безопасности

PSA определяет три режима для управления обработкой нарушений политики:

Режим	Поведение	Сценарий использования
Enforce	Запрещает создание/изменение Pod, не соответствующих требованиям.	Производственные среды, требующие строгого соблюдения безопасности.
Audit	Позволяет создавать Pod, но регистрирует нарушения в аудит-логах.	Мониторинг и анализ инцидентов безопасности без блокировки рабочих нагрузок.

Режим	Поведение	Сценарий использования
Warn	Позволяет создавать Pod, но возвращает клиенту предупреждения о нарушениях.	Тестовые среды или переходные этапы для корректировки политик.

Ключевые замечания:

- Режим **Enforce** действует только на Pod (например, отклоняет Pod, но разрешает ресурсы, не являющиеся Pod, такие как Deployments).
- Режимы **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 рабочих нагрузок (например, системных компонентов).	Нет проверки полей `securityContext`.
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` - Удалённые 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

```bash
# 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:

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

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 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 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 вы можете увидеть 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 находится в пределах 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 помощью следующей YAML-конфигурации:

```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 ошибкой.

1. Создайте Pod в namespace с помощью следующей YAML-конфигурации:

```
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 ошибкой с сообщением, что указанные UID/GID находятся вне назначенного диапазона.

Коэффициент Overcommit

Содержание

Понимание коэффициента overcommit ресурсов в Namespace

Alauda Container Platform позволяет задавать 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 заданные пользователем лимиты и запросы контейнеров остаются в разумных пределах, повышая общую эффективность использования ресурсов кластера.

Ключевые понятия

- Лимиты: максимальное количество ресурса, которое контейнер может использовать. Превышение лимитов может привести к троттлингу (CPU) или завершению процесса (память).

- Запросы: гарантированный минимум ресурса, необходимый контейнеру. Kubernetes планирует размещение контейнеров на основе этих запросов.
- Коэффициент Overcommit: Лимиты / Запросы. Эта настройка определяет допустимый диапазон этого соотношения внутри namespace, балансируя между гарантиями ресурсов и предотвращением их чрезмерного потребления.

Основные возможности

- Повышение плотности использования ресурсов и стабильности приложений в namespace за счёт установки подходящего коэффициента overcommit для управления 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 overcommit кластера; 0 - отсутствие ограничения.
  memory: 4 # Отсутствие этого поля означает наследование коэффициента overcommit кластера; 0 - отсутствие ограничения.
status:
  clusterCPU: 2 # Коэффициент overcommit кластера
  clusterMemory: 3
```

Создание коэффициента overcommit с помощью CLI

```
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 **способ 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, то namespace по умолчанию будет 4. • Запросы контейнера = лимит / коэффициент кластера. • Если лимит не установлен, используется квота контейнера по умолчанию в namespace.
Custom	• Установить специфичный для namespace коэффициент (целое число > 1). • Пример: коэффициент кластера = 4, коэффициент namespace = 2 → запросы = лимит / 2. • Оставьте пустым, чтобы отключить oversubscription для namespace.

1. 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 прав.

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 **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. На вкладке **Namespace Members** нажмите **Import Members**.
5. Следуйте нижеописанным шагам для импорта 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 в качестве 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 в качестве участников пространства имён.
     1. Отметьте флажком слева от имени пользователя/отображаемого имени 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 добавлять 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 тех, кто ещё не входил.

Предварительные условия

На платформе добавлен IDP типа **OICD**.

Процедура

1. В **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. На вкладке **Namespace Members** нажмите **Add Member**.
5. В поле ввода **Username** введите имя пользователя существующего аккаунта сторонней платформы, поддерживаемой 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 такого пользователя 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 прав в пространстве имён.

Процедура

1. В **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. На 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** > **Список Namespace**.
2. Нажмите на нужное ***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** > **Список Namespace**.
2. Нажмите на нужное ***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** > **Список Namespace**.
2. Нажмите на нужное ***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 Namespaces

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

Содержание

Удаление Namespaces

Удалить Namespace: Навсегда удаляет namespace и все ресурсы внутри него (например, Pods, Services, ConfigMaps). Это действие необратимо и освобождает выделенные квоты ресурсов.

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

Исключение Namespaces

Исключить Namespace: Исключение namespace из текущего проекта без удаления его ресурсов. Namespace остается в кластере и может быть импортирован в другие проекты через Import Namespace.

NOTE

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

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

Menu

Создание приложений

Создание приложений из образа

Создание приложений из Chart

Создание приложений из YAML

Создание приложений из кода

Creating applications from Operator Backed

Создание приложений с использованием CLI

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 загружать 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 **Already Integrated**. **Integration Project Name**, например, образы (docker-registry-projectname), включают имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

1. Следуйте инструкциям ниже 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: • **Deployment**: Для подробного описания параметров см. Creating Deployment. • **DaemonSet**: Для подробного описания параметров см. Creating DaemonSet.

Параметры	Описание
	• **StatefulSet**: Для подробного описания параметров см. Creating StatefulSet.
Replicas	Определяет желаемое количество реплик Pod в Deployment (по умолчанию: `1`). Настраивайте в зависимости от требований workload.
More > **Update Strategy**	Настраивает стратегию `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**: • `maxSurge` и `maxUnavailable` не могут одновременно быть `0` или `0%`.

Параметры	Описание
	• 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 развертывании.

Workload 2 — Настройка Pod

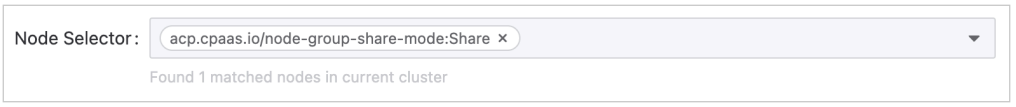
Примечание: В кластерах с разной архитектурой, при развертывании образов однородной архитектуры, убедитесь, что настроены корректные Node Affinity Rules для планирования Pod.

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 другие. Для деталей реализации см. Storage Volume Mounting Instructions.
Image Credential	Требуется **только** при загрузке образов из сторонних реестров (через ручной ввод URL образа). **Примечание**: Образы из интегрированного реестра платформы автоматически наследуют связанные секреты.

Параметры	Описание
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.

1. **Node Affinity Rules**

Параметры	Описание
More > **Node Selector**	Ограничивает планирование Pod на узлы с определёнными метками (например, `kubernetes.io/os: linux`). 
More > **Affinity**	Определяет тонкие правила планирования на основе существующих Pod. **Типы Pod Affinity**: • **Inter-Pod Affinity**: Планирование новых Pod на узлы, где размещены определённые Pod (в пределах одной топологической области). • **Inter-Pod Anti-affinity**: Запрет совместного размещения новых Pod с определёнными Pod. **Режимы применения**: • **RequiredDuringSchedulingIgnoredDuringExecution**: Pod планируются *только* если правила выполняются. • **PreferredDuringSchedulingIgnoredDuringExecution**: Предпочтение узлам, удовлетворяющим правилам, но допускаются исключения.

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

1. **Настройка сети**

- 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 предопределённого пула подсетей. Если не указано, используется подсеть по умолчанию для namespace.
Static IP Address	Привязка постоянных IP-адресов к Pod: • Несколько Pod в разных Deployment могут претендовать на один IP, но одновременно использовать его может только один Pod. • **Критично**: Количество статических IP должно быть ≥ числу 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 ≥ числу реплик Pod.

Workload 3 — Настройка контейнеров

1. В разделе **Container** следуйте инструкциям для настройки соответствующей информации.

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 namespace**: • **Без коэффициента overcommit**: Если существуют квоты ресурсов namespace: Requests/limits наследуют значения по умолчанию namespace (можно изменить). Без квот namespace: Нет значений по умолчанию; настраивается Request. • **С коэффициентом overcommit**: Requests рассчитываются автоматически как `Limits / Overcommit ratio` (неизменяемо). **Ограничения**:

Параметры	Описание
	• 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 постоянного 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**: • `protocol`: TCP/UDP • `Port`: Открываемый порт (например, `6379`)

Параметры	Описание
	• `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 (исключение из endpoints при сбое) См. Health Check Parameters.
More > **Log File**	Настройка путей логов: - По умолчанию: сбор `stdout` - Шаблоны файлов: например, `/var/log/*.log` **Требования**:

Параметры	Описание
	• Драйвер хранения `overlay2`: поддерживается по умолчанию • `devicemapper`: вручную монтировать EmptyDir в каталог логов • Узлы Windows: обеспечить монтирование родительского каталога (например, `c:/a` для `c:/a/b/c/*.log`)
More > **Exclude Log File**	Исключение конкретных 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:

   1.1. Запускается перед контейнерами приложения (последовательное выполнение).

   1.2. Освобождает ресурсы после завершения.

   1.3. Удаление разрешено, если:

   - Pod содержит >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 в вашем кластере через единую внешнюю точку доступа, даже если workload распределён по нескольким backend. Для подробного описания параметров см. Creating Services. **Примечание**: Префикс имени по умолчанию для внутреннего маршрута, созданного в приложении, — имя вычислительного компонента. Если тип вычислительного компонента (режим развертывания) — StatefulSet, рекомендуется не менять имя внутреннего маршрута (имя workload), иначе это может привести к проблемам с доступом к workload.

Процедура 3 — Ingress

Параметры	Описание
Ingress	Kubernetes **Ingress**, позволяет сделать ваш HTTP (или HTTPS) сетевой сервис доступным с помощью протокольно-ориентированной конфигурации, понимающей такие веб-концепции, 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** в приложении, должен быть ресурсом, созданным в текущем приложении. При этом убедитесь, что **Service** связан с workload в приложении, иначе обнаружение сервиса и доступ к workload будут невозможны.

1. 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 существующее количество реплик и конфигурацию развертывания. • **Force Update**: Запускает полное развертывание приложения с использованием стратегии обновления каждого компонента. 1. **Сценарии использования**: • Пакетные изменения конфигурации, 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 при массовых перезапусках. • Не рекомендуется использовать в продуктивных средах без проверки непрерывности бизнеса. • **Сетевые последствия**: • Удаление правил Ingress: Внешний доступ остаётся доступным через `LB_IP:NodePort`, если: 1) Service типа LoadBalancer использует стандартные порты. 2) Сохранившиеся правила маршрутизации ссылаются на

Операция	Описание
	компоненты приложения. Полное прекращение внешнего доступа требует удаления Service. • Удаление Service: Необратимая потеря сетевой доступности компонентов приложения. Связанные правила Ingress перестают работать, несмотря на сохранение объектов API.
Delete	• **Каскадное удаление**: 1. Удаляет все дочерние ресурсы, включая Deployments, Services и правила Ingress. 2. Persistent Volume Claims (PVC) обрабатываются согласно политике хранения, определённой в StorageClass. • **Проверочный список перед удалением**: 1. Убедитесь, что через связанные Services не идёт активный трафик. 2. Подтвердите завершение резервного копирования данных для stateful компонентов. 3. Проверьте зависимости ресурсов с помощью `kubectl describe ownerReferences`.

Справочная информация

Инструкции по монтированию томов хранения

Тип	Назначение
Persistent Volume Claim	Привязывает существующий PVC для запроса постоянного хранилища. **Примечание**: Выбираются только привязанные PVC (с

Тип	Назначение
	ассоциированным PV). Непривязанные PVC приведут к ошибкам создания pod.
ConfigMap	Монтирует полные или частичные данные ConfigMap как файлы: • Полный ConfigMap: создаёт файлы с именами ключей под путём монтирования • Выбор подпути: монтирует конкретный ключ (например, `my.cnf`)
Secret	Монтирует полные или частичные данные Secret как файлы: • 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 запуске pod - Удаляется при удалении pod **Сценарий использования**: обмен файлами между контейнерами, временное хранение данных. См. EmptyDir

Тип	Назначение
Host Path	Монтирует каталог хост-машины (должен 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`.
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`: отключает действия при ошибках - По умолчанию: `5` неудач → перезапуск контейнера.

Параметры, специфичные для протокола

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`).

Параметр	Применимые протоколы	Описание
HTTP Headers	HTTP/HTTPS	Пользовательские заголовки (добавьте пары 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

Использование 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 контроля версий. 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 узлов, 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, будьте терпеливы. В случае неудачи следуйте подсказкам интерфейса для завершения операции.

Справка по анализу статуса

Нажмите на ***Имя приложения***, чтобы отобразить подробный анализ статуса Chart в информации о деталях.

Тип	Причина
Initialized	Отражает статус загрузки шаблона Chart. • **True**: Шаблон Chart успешно загружен.

Тип	Причина

	• **False**: Загрузка шаблона Chart не удалась; конкретную причину ошибки можно посмотреть в столбце сообщения.
	  • `ChartLoadFailed` : Ошибка загрузки шаблона Chart.
	  • `InitializeFailed` : Исключение в процессе инициализации до загрузки Chart.

Validated	Отражает статус проверки прав пользователя, зависимостей и других валидаций шаблона Chart.
	• **True**: Все проверки прошли успешно.
	• **False**: Есть проверки, которые не прошли; конкретную причину ошибки можно посмотреть в столбце сообщения.
	  • `DependenciesCheckFailed` : Ошибка проверки зависимостей Chart.
	  • `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	Отражает 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

Когда 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 `docker pull`.

Процедура

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.

```yaml
# 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
```

Создание приложений из кода

Создание приложения из кода реализовано с использованием технологии Source to Image (S2I). S2I — это автоматизированная платформа для сборки контейнерных образов непосредственно из исходного кода. Такой подход стандартизирует и автоматизирует процесс сборки приложений, позволяя разработчикам сосредоточиться на разработке исходного кода, не беспокоясь 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, с рабочими процессами, управляемыми Operator Lifecycle Manager (OLM). Это включает ключевые процессы, такие как установка, обновления, разрешение зависимостей и контроль доступа.

Основные возможности

1. **Автоматизация сложных операций**: Operators преодолевают ограничения нативных ресурсов Kubernetes (например, Deployment, StatefulSet) для решения задач управления stateful-приложениями, включая распределённую координацию, постоянное хранилище и версионированные rolling updates. 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**:

   - Rolling Updates и Rollback: 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 корпоративной безопасности.

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 паттерны, широко используемые 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 Operator, содержащее имя, версию, предоставляемые API, необходимые разрешения, стратегию установки и подробную информацию о жизненном цикле.

3. InstallPlan ↗: Фактический план выполнения установки Operator, автоматически создаваемый OLM на основе Subscription и CSV, с детализацией шагов 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 управляемый Operator-ом экземпляр приложения 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 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 использованием 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 endpoints, и `kubectl` фактически преобразует команды CLI в соответствующие API-запросы для управления ресурсами кластера и рабочими нагрузками приложений (Deployments, StatefulSets и др.).

Инструмент CLI облегчает развертывание приложений, интеллектуально интерпретируя входные артефакты (образы, Chart и т. п.) и создавая 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, и 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 ↗

Menu

Эксплуатация и сопровождение приложений

Развертывание приложений

Установка Alauda Container Platform Argo Rollouts

Предварительные требования

Установка Alauda Container Platform Argo Rollouts

Application Blue Green Deployment

Требования

Процедура

Application Canary Deployment

Предварительные требования

Процедура

Описание статуса

Описание статуса

Приложения

KEDA (Kubernetes Event-driven Autoscaling)

KEDA Overview

Введение

Преимущества

Как работает KEDA

Установка KEDA

Предварительные требования

Установка через командную строку

Установка через веб-консоль

Проверка

Дополнительные сценарии

Удаление оператора KEDA

Как сделать

Настройка HPA

Настройка HPA

Понимание Horizontal Pod Autoscalers

Предварительные требования

Создание Horizontal Pod Autoscaler

Правила расчёта

Запуск и остановка приложений

Запуск и остановка приложений

Запуск приложения

Остановка приложения

Настройка VerticalPodAutoscaler (VPA)

Настройка VerticalPodAutoscaler (VPA)

Понимание VerticalPodAutoscalers

Предварительные требования

Создание VerticalPodAutoscaler

Последующие действия

Настройка CronHPA

Настройка CronHPA

Понимание Cron Horizontal Pod Autoscalers

Предварительные требования

Создание Cron Horizontal Pod Autoscaler

Объяснение правил расписания

Обновление приложений

Обновление приложений

Импорт ресурсов

Удаление/пакетное удаление ресурсов

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 состояния

Управление версиями приложений

Управление версиями приложений

Создание снимка версии

Откат к исторической версии

Удаление приложений

Удаление приложений

Проверки состояния

Проверки состояния

Понимание проверок состояния

Пример YAML файла

Параметры конфигурации проверок состояния через веб-консоль

Устранение неполадок с провалами проб

Menu

Развертывание приложений

Установка Alauda Container Platform Argo Rollouts

Содержание

Предварительные требования

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

INFO

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

Установка Alauda Container Platform Argo Rollouts

Процедура

1. Нажмите **Marketplace** > **Cluster Plugins**, чтобы перейти на страницу списка **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**.

Application Blue Green Deployment

В современном мире разработки программного обеспечения развертывание новых версий приложений является важной частью цикла разработки. Однако внедрение обновлений в продуктивные среды может быть рискованным процессом, так как даже небольшие проблемы могут привести к значительным простоям и потерям дохода. Стратегия развертывания Blue-Green 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 продуктивная среда, где в данный момент работает живая версия приложения, а зелёная — непроизводственная среда, куда разворачиваются новые версии приложения.

Когда новая версия приложения готова к развертыванию, она разворачивается в зелёной среде. После успешного развертывания и тестирования трафик переключается на зелёную среду, которая становится новой продуктивной средой. Синяя среда при этом становится непроизводственной, где можно разворачивать 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 просто, поскольку синяя среда остаётся доступной.
- Снижение рисков: Использование стратегии Blue-Green значительно снижает риски при развертывании новых версий, так как новая версия разворачивается и тестируется в зелёной среде до переключения трафика с синей. Это позволяет провести тщательное тестирование и уменьшить вероятность проблем в продуктиве.

- Повышение надёжности: Благодаря наличию синей среды, которая всегда доступна, любые проблемы с зелёной средой можно быстро выявить и устранить без влияния на пользователей.
- Гибкость: Стратегия Blue-Green предоставляет гибкость в процессе 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, предоставляющий 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 в процессе обновления.

С помощью Argo Rollouts вы можете автоматизировать развертывания Blue Green в 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. Развёртывание новой версии в зелёной среде.
4. Проверку и тестирование новой версии.
5. Продвижение зелёной среды в продуктив путём переключения трафика.

Такой подход минимизирует простои и обеспечивает контролируемое, безопасное развертывание.

> **Ключевые понятия:**
>
> - **Rollout**: определение пользовательского ресурса (CRD) в Kubernetes, заменяющее стандартные ресурсы Deployment и позволяющее управлять развертыванием 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 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 обращаются пользователи. Создайте Deployment Kubernetes 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 API Kubernetes, 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` : Docker-образ для запуска.
- `containers.ports.containerPort` : порт, открываемый контейнером.

Примените конфигурацию с помощью `kubectl` :

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

Это создаст продуктивную среду.

В качестве альтернативы можно использовать helm chart для создания Deployment 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 API Kubernetes 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 конфигурацию:

```
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 управляться этим Rollout. Должен совпадать с метками шаблона пода.
- `workloadRef`: ссылка на workload и стратегия масштабирования после миграции на Rollout.
  - `scaleDown`: указывает, будет ли Deployment масштабироваться вниз после успешного Rollout. Возможные значения:
    - "never": Deployment не масштабируется вниз.
    - "onsuccess": Deployment масштабируется вниз после того, как Rollout становится здоровым.
    - "progressively": Deployment масштабируется вниз по мере масштабирования Rollout вверх. Если Rollout неудачен, Deployment масштабируется обратно.
- `strategy`: стратегия развертывания, поддерживает `BlueGreen` и `Canary`.

- `blueGreen`: определение стратегии BlueGreen.
  - `activeService`: сервис, который обновляется 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 конфигурацию:

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

Это настроит Rollout с использованием стратегии BlueGreen.

6 Проверка Rollout

После создания `Rollout` Argo Rollouts создаст новый ReplicaSet с тем же шаблоном, что и 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     ✔ Healthy  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 конфигурацию:

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

Это подготовит новую версию приложения для тестирования.

Rollout создаст новый ReplicaSet для управления зелёными подами, при этом трафик по-прежнему направляется 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 указывает 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. Выполните 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     ✔ Healthy      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  • ScaledDown   3h1m
      ├──□ rollout-bluegreen-595d4567cc-mc769  Pod         ◌ Terminating  3h
ready:1/1
      └──□ rollout-bluegreen-595d4567cc-zdc5x  Pod         ◌ Terminating  3h
ready:1/1
```

Если активный образ обновлён на `hello:1.23.2`, а синий ReplicaSet масштабирован до 0, значит Rollout завершён успешно.

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 прогрессивной доставки, 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 Automated Rollbacks. Интегрируется со стеками 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 управлять продвинутыми стратегиями развертывания, такими как blue-green и 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 canary.

Преимущества канареечных развертываний

- **Снижение рисков**: развертывая изменения сначала на небольшой части серверов, можно выявить и устранить проблемы до полного выпуска, минимизируя влияние на пользователей.

- **Пошаговый выпуск**: такой подход позволяет постепенно вводить новые функции, что помогает эффективно контролировать производительность и отзывы пользователей.
- **Обратная связь в реальном времени**: канареечные развертывания дают мгновенную информацию о производительности и стабильности новых релизов в реальных условиях.
- **Гибкость**: можно корректировать процесс развертывания на основе метрик производительности, позволяя динамически 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 canary для развертывания и управления трафиком через Gateway API Plugin. В ACP можно использовать ALB в качестве Gateway API Provider для реализации управления трафиком для Argo Rollouts.

Содержание

Предварительные требования

1. Argo Rollouts с установленным Gateway API plugin в кластере.
2. kubectl плагин Argo Rollouts (установка доступна здесь ↗).
3. Проект для создания в нем namespace.
4. ALB, развернутый в кластере и выделенный проекту.
5. Namespace в кластере, где 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` : Docker-образ для запуска.
- `containers.ports.containerPort` : порт, открываемый контейнером.

Примените конфигурацию с помощью `kubectl` :

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

Это настроит продуктивную среду.

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

2 Создание стабильного сервиса

Создайте Kubernetes `Service`, который будет открывать стабильный deployment. Этот сервис будет направлять 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 направляется 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 открывать канареечный deployment. Этот сервис направит 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 направляется 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

Используйте `example.com` в качестве домена для доступа к сервису, создайте gateway для экспонирования сервиса с этим доменом:

```yaml
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, получите его 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 домен в вашем DNS-сервере так, чтобы он 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
```

Должен возвращаться 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: /
```

Выполните команду:

```
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, чьи поды 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": по мере масштабирования Rollout Deployment масштабируется вниз. Если Rollout неудачен, Deployment масштабируется обратно.
- `strategy` : стратегия развертывания, поддерживает `BlueGreen` и `Canary` .
- `canary` : определение стратегии `Canary` .
  - `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.
    - `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 вес установится в 100, и весь трафик пойдет 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  32s  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 новым образом (например, `hello:1.23.2`). Используйте команду:

```
kubectl patch deployment web -p '{"spec":{"template":{"spec":{"containers":
[{"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   95s
├──# revision:2
│  └──⧉ rollout-canary-5898765588          ReplicaSet  ✔ Healthy  46s  canary
│     └──□ rollout-canary-5898765588-ls5jk  Pod         ✔ Running  45s
ready:1/1
└──# revision:1
   └──⧉ rollout-canary-5c9d79697b          ReplicaSet  ✔ Healthy  95s  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 получить разные ответы от URL.

11 Продвижение Rollout

Когда канареечная версия протестирована и готова, можно 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 stable `Images` обновлен до `hello:1.23.2`, а ReplicaSet ревизии 1 масштабирован до 0, значит rollout завершен.

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

12 Прерывание Rollout (опционально)

Если во время rollout канареечная версия вызывает проблемы, можно 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        AGE  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 завершилось с ошибкой.

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

Развертывание

- Running: Все Pods работают в нормальном режиме.
- Processing: Есть Pods, которые не находятся в состоянии Running.
- Stopped: Все Pods остановлены.
- Failed: Развертывание завершилось с ошибкой.

Menu

KEDA (Kubernetes Event-driven Autoscaling)

KEDA Overview

Установка KEDA

Как сделать

Интеграция ACP Monitoring с плагином Prometheus

Приостановка автоскейлинга в KEDA

KEDA Overview

Содержание

Введение

KEDA — это основанный на Kubernetes автоскейлер с событийным управлением. Home Page ↗. С помощью KEDA вы можете масштабировать любой контейнер в Kubernetes в зависимости от количества событий, которые необходимо обработать.

KEDA — это специализированный и легковесный компонент, который можно добавить в любой кластер Kubernetes. KEDA работает вместе со стандартными компонентами Kubernetes, такими как Horizontal Pod Autoscaler ↗, и может расширять функциональность без перезаписи или дублирования. С KEDA вы можете явно указать приложения, для которых хотите использовать масштабирование на основе событий, при этом другие приложения продолжают работать. Это делает KEDA гибким и безопасным решением для работы вместе с любым количеством других приложений или фреймворков Kubernetes.

Подробности смотрите в официальной документации: Keda Documentation ↗

Преимущества

Основные преимущества KEDA:

- **Простое автоскейлирование:** Обеспечьте расширенные возможности масштабирования для любой нагрузки в вашем кластере Kubernetes.
- **Событийное управление:** Интеллектуальное масштабирование ваших событийно-ориентированных приложений.
- **Встроенные скейлеры:** Каталог из более чем 70 встроенных скейлеров для различных облачных платформ, баз данных, систем обмена сообщениями, телеметрии, CI/CD и других.
- **Поддержка различных типов нагрузок:** Поддержка различных типов нагрузок, таких как deployments, jobs и пользовательские ресурсы с подресурсом **/scale**.
- **Снижение воздействия на окружающую среду:** Создавайте устойчивые платформы, оптимизируя планирование нагрузки и масштабирование до нуля.
- **Расширяемость:** Используйте собственные или поддерживаемые сообществом скейлеры.
- **Независимость от поставщика:** Поддержка триггеров для различных облачных провайдеров и продуктов.
- **Поддержка Azure Functions:** Запускайте и масштабируйте ваши Azure Functions на Kubernetes в производственных нагрузках.

Как работает KEDA

KEDA отслеживает внешние источники событий и регулирует ресурсы вашего приложения в зависимости от спроса. Его основные компоненты работают вместе, чтобы это обеспечить:

1. **KEDA Operator** следит за источниками событий и изменяет количество экземпляров приложения вверх или вниз в зависимости от спроса.

2. **Metrics Server** предоставляет внешние метрики для HPA Kubernetes, чтобы он мог принимать решения о масштабировании.

3. **Scalers** подключаются к источникам событий, таким как очереди сообщений или базы данных, получая данные о текущей нагрузке или использовании.

4. **Custom Resource Definitions (CRDs)** определяют, как ваши приложения должны масштабироваться на основе триггеров, таких как длина очереди или скорость запросов API.

Проще говоря, KEDA слушает, что происходит вне Kubernetes, получает необходимые данные и масштабирует ваши приложения соответственно. Это эффективно и хорошо интегрируется с Kubernetes для динамического управления масштабированием.

KEDA Custom Resource Definitions (CRDs)

KEDA использует **Custom Resource Definitions (CRDs)** для управления поведением масштабирования:

- **ScaledObject**: Связывает ваше приложение (например, Deployment или StatefulSet) с внешним источником событий, определяя правила масштабирования.
- **ScaledJob**: Обрабатывает задачи пакетной обработки, масштабируя Jobs на основе внешних метрик.
- **TriggerAuthentication**: Обеспечивает безопасный доступ к источникам событий, поддерживая методы, такие как переменные окружения или облачные учетные данные.

Эти CRD дают вам контроль над масштабированием, сохраняя безопасность и отзывчивость приложений к изменению спроса.

Пример ScaledObject:

Следующий пример нацелен на использование CPU всего пода. Если в поде несколько контейнеров, будет учитываться сумма всех контейнеров.

```yaml
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: <your-namespace>
spec:
  scaleTargetRef:
    name: <your-deployment>
  triggers:
  - type: cpu
    metricType: Utilization # Allowed types are 'Utilization' or 'AverageValue'
    metadata:
      value: "50"
```

Установка KEDA

Содержание

Предварительные требования

KEDA — это инструмент, который помогает Kubernetes масштабировать приложения на основе реальных событий. С помощью KEDA вы можете автоматически регулировать количество контейнеров в зависимости от нагрузки — например, количества сообщений в очереди или входящих запросов.

1. **Скачайте** установочный пакет **KEDA** из **Alauda Cloud**.

2. **Загрузите** установочный пакет с помощью механизма Upload Packages.

> **INFO**
>
> Upload Packages:
>
> **Administrator** > **Marketplace** > **Upload Packages** страница.
>
> Нажмите **Help Document** справа, чтобы получить инструкции по публикации оператора в кластере. Для подробностей обратитесь к CLI.

Установка через командную строку

Установка оператора KEDA

Создайте namespace для оператора KEDA, если он не существует:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: "keda"
EOF
```

Выполните следующую команду для установки оператора KEDA в целевой кластер:

```
kubectl apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  annotations:
    cpaas.io/target-namespaces: ""
  labels:
    catalog: platform
  name: keda
  namespace: keda
spec:
  channel: stable
  installPlanApproval: Automatic
  name: keda
  source: custom
  sourceNamespace: cpaas-system
  startingCSV: keda.v2.17.2
EOF
```

Параметры конфигурации:

Параметр	Рекомендуемая конфигурация
metadata.name	`keda` : имя Subscription установлено как **keda**.
metadata.namespace	`keda` : namespace Subscription установлен как **keda**.
spec.channel	`stable` : по умолчанию выбран канал **stable**.
spec.installPlanApproval	`Automatic` : действие **Upgrade** будет выполняться автоматически.
spec.name	`keda` : имя пакета оператора, должно быть **keda**.
spec.source	`custom` : источник каталога оператора keda, должен быть **custom**.
spec.sourceNamespace	`cpaas-system` : namespace источника каталога, должен быть **cpaas-system**.
spec.startingCSV	`keda.v2.17.2` : имя стартового CSV оператора keda.

Создание экземпляра KedaController

Создайте ресурс KedaController с именем keda в namespace keda:

```
kubectl apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: KedaController
metadata:
  name: keda
  namespace: keda
spec:
  admissionWebhooks:
    logEncoder: console
    logLevel: info
  metricsServer:
    logLevel: "0"
  operator:
    logEncoder: console
    logLevel: info
  serviceAccount: null
  watchNamespace: ""
EOF
```

Установка через веб-консоль

Установка оператора KEDA

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace** > **OperatorHub**.
3. Найдите оператора **KEDA**, нажмите **Install** и перейдите на страницу **Install**.

Параметры конфигурации:

Параметр	Рекомендуемая конфигурация
Channel	`stable`: по умолчанию выбран канал **stable**.

Параметр	Рекомендуемая конфигурация
Version	Выберите последнюю версию.
Installation Mode	`Cluster`: один оператор используется во всех namespace кластера для создания и управления экземплярами, что снижает использование ресурсов.
Installation Location	`Recommended`: будет создан автоматически, если не существует.
Upgrade Strategy	Выберите `Auto`. • действие **Upgrade** будет выполняться автоматически.

1. На странице **Install** выберите конфигурацию по умолчанию, нажмите **Install** и завершите установку оператора **KEDA**.

Создание экземпляра KedaController

1. Нажмите **Marketplace** > **OperatorHub**.
2. Найдите установленного оператора **KEDA**, перейдите в раздел **All Instances**.
3. Нажмите кнопку **Create Instance** и выберите карточку **KedaController** в области ресурсов.
4. На странице настройки параметров экземпляра можно использовать конфигурацию по умолчанию, если нет специальных требований.
5. Нажмите **Create**.

Проверка

После успешного создания экземпляра подождите примерно 20 минут, затем проверьте, запущены ли компоненты KEDA командой:

```
kubectl get pods -n keda
```

Дополнительные сценарии

Интеграция с ACP Log Collector

- Убедитесь, что **ACP Log Collector Plugin** установлен в целевом кластере. См. ACP Log Collector Plugin Install.
- Включите переключатель логирования **Platform** при установке **ACP Log Collector Plugin**.
- Используйте следующую команду для добавления метки в namespace **keda**:

```
kubectl label namespace keda cpaas.io/product=Container-Platform --overwrite
```

Удаление оператора KEDA

Удаление экземпляра KedaController

```
kubectl delete kedacontroller keda -n keda
```

Удаление оператора KEDA через CLI

```
kubectl delete subscription keda -n keda
```

Удаление оператора KEDA через веб-консоль

Чтобы удалить оператор KEDA, перейдите в **Marketplace** > **OperatorHub**, выберите установленного оператора **KEDA** и нажмите **Uninstall**.

Menu

Как сделать

Интеграция ACP Monitoring с плагином Prometheus

Предварительные требования

Процедура

Проверка

Приостановка автоскейлинга в KEDA

Процедура

Масштабирование до нуля

Проверка

Интеграция ACP Monitoring с плагином Prometheus

> В этом руководстве описывается, как настроить интеграцию с **ACP Monitoring с плагином Prometheus** для автоматического масштабирования приложения на основе метрик Prometheus.

Содержание

Предварительные требования

Перед использованием данной функции убедитесь, что:

- Установлен ACP Monitoring с плагином Prometheus
- Получен URL конечной точки Prometheus и secretName для текущего кластера Kubernetes:

```
PrometheusEndpoint=$(kubectl get feature monitoring -o
jsonpath='{.spec.accessInfo.database.address}')
```

- Получен секрет Prometheus для текущего кластера Kubernetes:

```
PrometheusSecret=$(kubectl get feature monitoring -o
jsonpath='{.spec.accessInfo.database.basicAuth.secretName}')
```

- Создан деплоймент с именем `<your-deployment>` в пространстве имён `<your-namespace>`.

Процедура

- Настройте секрет аутентификации Prometheus в пространстве имён **keda**.

Шаги для копирования секрета из cpaas-system в пространство имён keda

```
# Получить данные аутентификации Prometheus
PrometheusUsername=$(kubectl get secret $PrometheusSecret -n cpaas-system -o
jsonpath='{.data.username}' | base64 -d)
PrometheusPassword=$(kubectl get secret $PrometheusSecret -n cpaas-system -o
jsonpath='{.data.password}' | base64 -d)

# создать секрет в пространстве имён keda
kubectl create secret generic $PrometheusSecret \
  -n keda \
  --from-literal=username=$PrometheusUsername \
  --from-literal=password=$PrometheusPassword
```

- Настройте аутентификацию KEDA для доступа к Prometheus с помощью **ClusterTriggerAuthentication**.

Для настройки учетных данных аутентификации, чтобы KEDA могла получить доступ к Prometheus, определите ресурс ClusterTriggerAuthentication, который ссылается на Secret с именем пользователя и паролем. Ниже приведён пример конфигурации:

```
kubectl apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: ClusterTriggerAuthentication
metadata:
  name: cluster-prometheus-auth
spec:
  secretTargetRef:
    - key: username
      name: $PrometheusSecret
      parameter: username
    - key: password
      name: $PrometheusSecret
      parameter: password
EOF
```

- Настройте автоматическое масштабирование для Kubernetes Deployment на основе метрик Prometheus с помощью **ScaledObject**.

Чтобы масштабировать Kubernetes Deployment на основе метрик Prometheus, определите ресурс **ScaledObject**, ссылающийся на настроенный ClusterTriggerAuthentication. Ниже приведён пример конфигурации:

```
kubectl apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prometheus-scaledobject
  namespace: <your-namespace>
spec:
  cooldownPeriod: 300          # Время в секундах ожидания перед масштабированием
вниз
  maxReplicaCount: 5           # Максимальное количество реплик
  minReplicaCount: 1           # Минимальное количество реплик (обратите внимание:
HPA может требовать минимум 1)
  pollingInterval: 30          # Интервал (в секундах) опроса метрик Prometheus
  scaleTargetRef:
    name: <your-deployment>    # Имя целевого Kubernetes Deployment
  triggers:
    - authenticationRef:
        kind: ClusterTriggerAuthentication
        name: cluster-prometheus-auth  # Ссылка на ClusterTriggerAuthentication
      metadata:
        authModes: basic       # Метод аутентификации (в данном случае basic auth)
        query:
sum(container_memory_working_set_bytes{container!="POD",container!="",namespace="<your-
namespace>",pod=~"<your-deployment-name>.*"})
        queryParameters: timeout=10s  # Дополнительные параметры запроса
(необязательно)
        serverAddress: $PrometheusEndpoint
        threshold: "1024000"   # Пороговое значение для масштабирования
        unsafeSsl: "true"      # Пропуск проверки SSL сертификата (не рекомендуется
для продакшена)
      type: prometheus         # Тип триггера
EOF
```

Проверка

Чтобы проверить, что ScaledObject масштабировал деплоймент, можно проверить количество реплик целевого деплоймента:

```
kubectl get deployment <your-deployment> -n <your-namespace>
```

Или можно использовать следующую команду для проверки количества подов:

```
kubectl get pods -n <your-namespace> -l <your-deployment-label-key>=<your-deployment-label-value>
```

Количество реплик должно увеличиваться или уменьшаться в зависимости от метрик, указанных в ScaledObject.
Если деплоймент масштабируется корректно, вы увидите, что количество подов изменилось до значения `maxReplicaCount`.

Другие скейлеры KEDA

Скейлеры KEDA могут как определять, нужно ли активировать или деактивировать деплоймент, так и предоставлять пользовательские метрики для конкретного источника событий.

KEDA поддерживает широкий спектр дополнительных скейлеров. Для получения подробной информации смотрите официальную документацию: KEDA Scalers↗.

Приостановка автоскейлинга в KEDA

KEDA позволяет временно приостанавливать автоскейлинг рабочих нагрузок, что полезно для:

- Обслуживания кластера.
- Избежания нехватки ресурсов за счёт масштабирования не критичных рабочих нагрузок вниз.

Содержание

Процедура

Немедленная пауза с текущим количеством реплик

Добавьте следующую аннотацию в определение вашего **ScaledObject**, чтобы приостановить масштабирование без изменения текущего количества реплик:

```
metadata:
  annotations:
    autoscaling.keda.sh/paused: "true"
```

Пауза после масштабирования до определённого количества реплик

Используйте эту аннотацию, чтобы масштабировать рабочую нагрузку до заданного количества реплик, а затем приостановить:

```
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "<number>"
```

Поведение при установке обеих аннотаций

Если указаны обе аннотации — **paused** и **paused-replicas**:

- KEDA масштабирует рабочую нагрузку до значения, определённого в **paused-replicas**.
- После этого автоскейлинг приостанавливается.

Возобновление автоскейлинга

Чтобы возобновить автоскейлинг:

- Удалите обе аннотации paused и paused-replicas из ScaledObject.
- Если использовалась только paused: "true", установите её в false:

```
metadata:
  annotations:
    autoscaling.keda.sh/paused: "false"
```

Масштабирование до нуля

Пример конфигурации ScaledObject:

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: example-scaledobject
  namespace: <your-namespace>
  annotations:
    autoscaling.keda.sh/paused-replicas: "0"  # Масштабировать до 0 реплик и
приостановить
```

Проверка

Чтобы проверить, что ScaledObject масштабировался до нуля, можно проверить количество реплик целевого deployment:

```
kubectl get deployment <your-deployment> -n <your-namespace>
```

Или проверить количество pod-ов в целевом deployment:

```
kubectl get pods -n <your-namespace> -l <your-deployment-label-key>=<your-deployment-
label-value>
```

Количество pod-ов должно быть равно нулю, что указывает на масштабирование deployment до нуля.

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, одновременно оптимизируя использование 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% при масштабировании вниз. Это обеспечивает большую стабильность при принятии решений HPA. Для использования этой функции необходимо настроить readiness проверки, 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, когда заданный порог CPU или памяти превышается.

HPA работает как управляющий цикл с периодом синхронизации по умолчанию 15 секунд. В течение этого периода 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 ресурсов контейнеров в каждом поде. 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
Использование CPU	Количество используемых ядер CPU. Может использоваться для вычисления процента от запрошенного CPU пода.
Использование памяти	Объем используемой памяти. Может использоваться для вычисления процента от запрошенной памяти пода.
Входящий сетевой трафик	Объем сетевого трафика, поступающего в под, измеряется в KiB/s.
Исходящий сетевой трафик	Объем сетевого трафика, исходящего из пода, измеряется в KiB/s.
Трафик чтения со хранилища	Объем данных, читаемых со хранилища, измеряется в KiB/s.
Трафик записи на хранилище	Объем данных, записываемых на хранилище, измеряется в KiB/s.

> **Важно**: Для автоскейлинга на основе памяти использование памяти должно пропорционально увеличиваться и уменьшаться в зависимости от количества реплик. В среднем:
>
> - Увеличение количества реплик должно приводить к общему снижению использования памяти (working set) на один под.
> - Уменьшение количества реплик должно приводить к общему увеличению использования памяти на один 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 Horizontal Pod Autoscaler

Использование CLI

Вы можете создать горизонтальный автоскейлер подов с помощью интерфейса командной строки, определив YAML-файл и используя команду `kubectl create`. Следующий пример показывает автоскейлинг для объекта Deployment. Изначально требуется 3 пода. Объект HPA увеличивает минимум до 5. Если использование CPU подов 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, который запускает 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 новое 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 нажмите **Workloads** > **Deployments**.
3. Кликните по ***Имя Deployment***.
4. Прокрутите вниз до области **Elastic Scaling** и нажмите **Update** справа.
5. Выберите **Horizontal Scaling** и заполните конфигурацию политики.

Параметр	**Описание**
Количество подов	После успешного создания деплоя необходимо оценить **Минимальное количество подов**, соответствующее известным и регулярным изменениям бизнес-объема, а

Параметр	Описание
	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 постоянно корректировать в процессе эксплуатации для удовлетворения бизнес-потребностей.
Политика триггера	Укажите **Метрики**, чувствительные к изменениям бизнеса, и их **Целевые пороги** для запуска действий масштабирования вверх или вниз. Например, если вы установите *CPU Utilization = 60%*, как только использование CPU отклонится от 60%, платформа начнет автоматически корректировать количество подов на основе недостаточного или избыточного распределения 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 .
Шаг масштабирования (альфа)	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 действий по уменьшению масштаба.

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 использование 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 типа 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:

```
{
      "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:

```
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, вычисленное из прямого значения

Алгоритм:

```
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; будьте осторожны.

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 установленными **Минимальным количеством подов** или **Максимальным количеством подов**.

- Целевое количество подов по одной политике: ceil[(сумма фактических значений метрик) / порог метрики]. Это означает, что сумма фактических значений метрик всех подов делится на порог метрики и округляется вверх до ближайшего целого числа. Например: если в данный момент 3 пода с использованием CPU 80%, 80% и 90%, а установлен порог CPU 60%, то согласно 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, такие как увеличение квоты подов namespace или добавление аппаратных ресурсов.

  - Если рассчитанное целевое количество подов (в предыдущем примере *5*) меньше количества подов, рассчитанного с учетом **Шага масштабирования вверх** (например, *10*), платформа масштабирует только до 5 подов.

- Целевое количество подов по нескольким политикам: выбирается 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)

Для как stateless, так и stateful приложений VerticalPodAutoscaler (VPA) автоматически рекомендует и при необходимости применяет более подходящие лимиты ресурсов CPU и памяти на основе ваших бизнес-требований, обеспечивая достаточное количество ресурсов для подов и повышая эффективность использования ресурсов кластера.

Содержание

Понимание VerticalPodAutoscalers

Вы можете создать VerticalPodAutoscaler для рекомендации или автоматического обновления запросов и лимитов ресурсов CPU и памяти для ваших подов на основе их исторических шаблонов использования.

После создания VerticalPodAutoscaler платформа начинает мониторить использование ресурсов CPU и памяти подами. Когда собирается достаточное количество данных, VerticalPodAutoscaler рассчитывает рекомендуемые значения ресурсов на основе наблюдаемых шаблонов использования. В зависимости от настроенного режима обновления VPA может либо автоматически применять эти рекомендации, либо просто предоставлять их для ручного применения.

VPA работает, анализируя использование ресурсов ваших подов с течением времени и делая рекомендации на основе этого анализа. Это помогает обеспечить подам необходимые ресурсы без избыточного выделения, что ведёт к более эффективному использованию ресурсов в кластере.

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, непрерывно мониторя использование ресурсов подами 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 может обеспечивать горизонтальное или вертикальное масштабирование подов. При наличии достаточных ресурсов эластичное масштабирование даёт хорошие результаты, но при нехватке ресурсов кластера это может привести к состоянию Pending у подов. Поэтому убедитесь, что в кластере достаточно ресурсов или установлены разумные квоты, либо настройте оповещения для мониторинга условий масштабирования.

Поддерживаемые функции

VerticalPodAutoscaler 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. Рекомендуется применять рекомендации в окна обслуживания для рабочих нагрузок в продакшене.

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**..
- В вашем кластере должен быть установлен кластерный плагин Vertical Pod Autoscaler от Alauda Container Platform.

Установка плагина Vertical Pod Autoscaler

Перед использованием VPA необходимо установить кластерный плагин 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.

Создание VerticalPodAutoscaler

Использование CLI

Вы можете создать VerticalPodAutoscaler через командную строку, 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 требуется корректировка ресурсов. 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. updateMode может принимать значения:

- Auto: Автоматически устанавливает запросы ресурсов при создании подов и обновляет текущие поды до рекомендуемых значений ресурсов. В настоящее время эквивалентно "Recreate". Этот режим может вызвать простой приложения. После поддержки обновлений ресурсов подов на месте режим "Auto" будет использовать этот механизм обновления.
- Recreate: Автоматически устанавливает запросы ресурсов при создании подов и эвакуирует текущие поды для обновления до рекомендуемых значений ресурсов. Не использует обновления на месте.

- Initial: Устанавливает запросы ресурсов только при создании подов, без последующих изменений.
- Off: Не изменяет запросы ресурсов подов автоматически, только предоставляет рекомендации в объекте VPA.

8 Политика ресурсов, позволяющая задавать конкретные 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 в 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 (частично):

```
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. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Workloads** > **Deployments**.
3. Кликните по ***Имя Deployment***.
4. Пролистайте вниз до области **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 использования 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 нехватке ресурсов кластера масштабирование может привести к состоянию Pending

Параметр	Описание
	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, без последующих изменений.
- `minReplicas: <number>` — Минимальное количество реплик. Обеспечивает наличие этого минимального количества подов при эвакуации подов Updater’ом. 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 для получения точных рекомендаций поды должны работать несколько дней.
- 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 ресурсов CPU и памяти для целевого контейнера можно просмотреть в области **Elastic Scaling**. В разделе **Containers** выберите вкладку целевого контейнера и нажмите иконку справа от **Resource Limits**, чтобы обновить лимиты ресурсов согласно рекомендуемым значениям.

Настройка CronHPA

Для бессостоянных приложений с периодическими колебаниями бизнес-активности 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 снизить использование ресурсов в часы низкой 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 низкой активности.

CronHPA постоянно сравнивает текущее время с заданными расписаниями. Когда наступает запланированное время, контроллер регулирует количество подов, чтобы оно соответствовало целевому количеству реплик, указанному для этого расписания. Если несколько расписаний срабатывают одновременно, платформа применит правило с более высоким приоритетом (то есть определённое 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, нажав в правом верхнем углу платформы ? > **Platform Health Status**..

Создание Cron Horizontal Pod Autoscaler

Использование CLI

Вы можете создать cron horizontal pod autoscaler с помощью 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 версию API 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

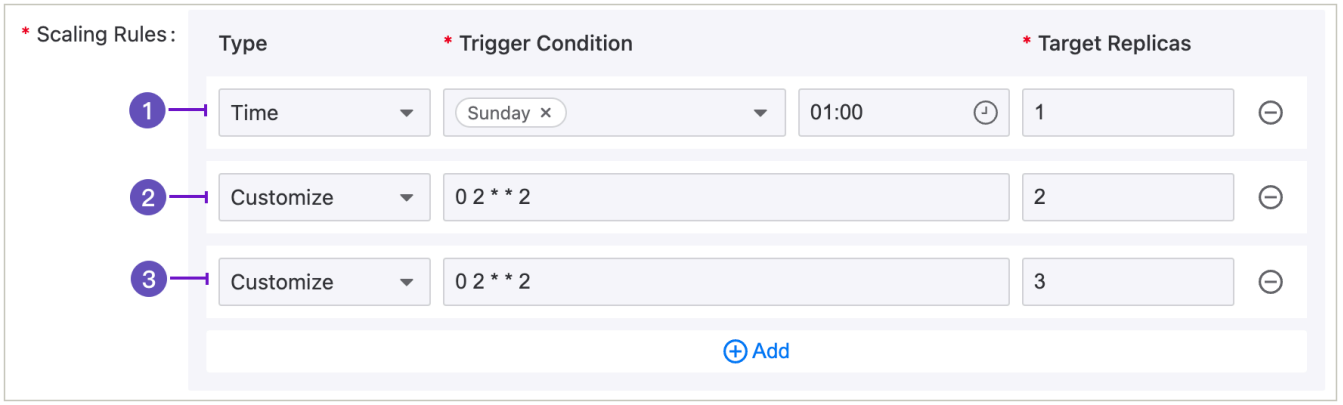
2. Примените YAML-файл для создания CronHPA:

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

Использование веб-консоли

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Workloads** > **Deployments**.
3. Нажмите на ***Имя деплоймента***.
4. Прокрутите вниз до раздела **Elastic Scaling** и нажмите справа **Update**.
5. Выберите **Scheduled Scaling** и настройте правила масштабирования. Если тип выбран как **Custom**, необходимо указать выражение Crontab для условия срабатывания в формате `минута час день месяц неделя`. Для подробного ознакомления 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 настройках часового пояса 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, хранилищем и конфигурациями, но не все ресурсы принадлежат приложению.

- Ресурсы, добавленные в процессе создания приложения или через обновления 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 работать 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 пространствами имён 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 информации о 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 пространстве имён, где находится приложение; ресурс может принадлежать только одному приложению.

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

3. Нажмите на ***Название приложения***.

4. Нажмите **Actions** > **Manage Resources**.

5. Справа от ресурса нажмите **Remove** для удаления; либо выберите несколько ресурсов сразу 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 ниже для настройки соответствующей информации.

   - Экспорт Helm Charts в репозиторий шаблонов с правами управления

     Примечание: Репозиторий шаблонов добавляется администратором платформы. Обратитесь к администратору платформы для получения действующего репозитория шаблонов типа **Chart** или **OCI Chart** с правами **Management**.

Параметр	Описание
Target Location	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.
Template Directory	Если выбран репозиторий шаблонов типа 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.

Параметр	Описание
Version	Номер версии шаблона приложения. Формат должен быть `v<Major>.<Minor>.<Patch>`. По умолчанию используется текущая версия приложения или текущая версия снимка.
Icon	Поддерживаются форматы изображений JPG, PNG и GIF, размер файла не более 500KB. Рекомендуемые размеры — 80*60 пикселей.
Description	Описание, которое будет отображаться в списке шаблонов приложений в каталоге приложений.
README	Файл описания. Поддерживается редактирование в формате Markdown и отображается на странице деталей шаблона приложения.
NOTES	Файл помощи шаблона. Поддерживается редактирование в обычном текстовом формате; после завершения шаблона развертывания он будет отображаться 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 скачан 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 использовать для прямого развертывания 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
Full YAML	По умолчанию опция **Preview Simplified YAML** не выбрана, отображается YAML-файл с **скрытыми полями managedFields**. Вы можете просмотреть и экспортировать его напрямую; также

Тип	Описание
	можно снять галочку с **Hide managedFields fields** для экспорта полного YAML-файла. **Примечание**: Полный YAML в основном используется для операций и устранения неполадок и не подходит для быстрого создания нативных приложений на платформе.
Simplified YAML	При выборе **Preview Simplified YAML** можно просмотреть и экспортировать упрощённый YAML-файл, который можно использовать для прямого развертывания 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 ниже для настройки соответствующей информации.

   - Экспорт YAML в репозиторий кода:

Параметр	Описание
Target Location	Выберите **Code Repository** для прямой синхронизации YAML-файла в указанный Git-репозиторий. Владелец проекта, назначенный для этого **Code Repository**, сможет напрямую использовать YAML-файл.
Integration Project Name	Название проекта интеграционного инструмента, назначенного или связанного с вашим проектом администратором платформы.
Repository Address	Адрес репозитория, назначенный для вашего использования в рамках интегрированного проекта инструмента.

Export Method	• **Existing Branch**: экспорт YAML приложения в выбранную ветку.
	• **New Branch**: создание новой ветки на основе выбранного **Branch/Tag/Commit ID** и экспорт YAML приложения в новую ветку.
	  • Если отмечена опция **Submit PR (Pull Request)**, платформа создаст новую ветку и отправит Pull Request.
	  • Если отмечена опция **Automatically delete source branch after merging PR**, исходная ветка будет

Параметр	Описание
	автоматически удалена после слияния PR в Git-репозитории.
File Path	Конкретное место сохранения файла в репозитории кода; можно также ввести путь к файлу, и платформа создаст новый путь в репозитории на основе введённого значения.
Commit Message	Заполните информацию о коммите для идентификации содержимого этого коммита.
Preview	Просмотр YAML-файла для отправки и сравнение изменений с существующим YAML в репозитории кода с цветовой дифференциацией.

- Экспорт файлов типа Kustomize локально для последующей ручной загрузки в репозиторий кода: выберите **Local** в качестве целевого расположения и формат файла **Kustomize** для экспорта файла конфигурации приложения типа Kustomize локально. 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 Созданию GitOps приложений 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 состояния

Нажмите на ***Template Application Name***, чтобы отобразить подробный анализ состояния 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 отсутствуют права 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 исторической версии. Для обновлений приложения, инициированных вне интерфейса, например, через вызовы API, 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 ***Название приложения***.

4. Во вкладке **Version Snapshot** нажмите **Create Version Snapshot**.
5. Заполните информацию и нажмите **Confirm**.

   Примечание: Вы также можете распространить приложение, что позволяет распространять снимок версии приложения в виде Chart, облегчая быстрое развертывание одного и того же приложения в нескольких кластерах и пространствах имён на платформе.

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 ***Название приложения***.
4. Во вкладке **Historical Versions** нажмите на ***Номер версии***.
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.

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 критическим 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 правильных 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 удалит Pod из списка Endpoint'ов Service до тех пор, пока проба не станет успешной.
- `startupProbe`: Специально проверяет, успешно ли запустилось приложение. Liveness и readiness пробы не будут выполняться, пока startup probe не пройдет успешно. Это очень 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.

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

- **Пример**:

```
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.

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

```
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 установлено.

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

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

Лучшие практики

- **Liveness** vs. **Readiness**:
  - **Liveness**: Если ваше приложение не отвечает, лучше перезапустить его. При сбое Kubernetes выполнит перезапуск.
  - **Readiness**: Если ваше 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 с длительным временем инициализации. Это предотвращает преждевременные перезапуски из-за сбоев Liveness Probe или проблемы с маршрутизацией трафика из-за сбоев Readiness Probe во время запуска.
- **Легковесные пробы**: Убедитесь, что ваши 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 действительно отражать состояние здоровья и готовности вашего приложения, а не просто факт работы процесса. Например, для веб-сервера проверяйте, может ли он обслуживать базовую страницу, а не просто открыт ли порт.
- **Настройка initialDelaySeconds**: Устанавливайте initialDelaySeconds так, чтобы дать приложению достаточно времени на запуск перед первой проверкой.
- **Настройка periodSeconds и failureThreshold**: Балансируйте необходимость быстрого обнаружения сбоев и избегайте ложных срабатываний. Слишком частые пробы или слишком низкий failureThreshold могут привести к ненужным перезапускам или состоянию «не готов».

- **Логи для отладки**: Обеспечьте, чтобы ваше приложение логировало понятные сообщения, связанные с вызовами эндпоинтов проверок и внутренним состоянием, для облегчения отладки 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 probe 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 старта Pod перед проверкой
            periodSeconds: 10 # Проверка каждые 10 секунд
            timeoutSeconds: 5 # Таймаут через 5 секунд
            failureThreshold: 3 # Считается нездоровым после 3 последовательных неудач
          readinessProbe:
            httpGet:
              path: /ready
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 10
            timeoutSeconds: 5
            failureThreshold: 3
```

Параметры конфигурации 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 неудаче - По умолчанию: `5` неудач → перезапуск контейнера.

Параметры, специфичные для протокола

Параметр	Применимые 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 (добавьте пары ключ-значение).
Command	EXEC	Команда для проверки, выполняемая в контейнере (например, `sh -c "curl -I localhost:8080 | grep OK"`). **Примечание**: Экранируйте специальные

Параметр	Применимые протоколы	Описание
		символы и проверьте работоспособность команды.

Устранение неполадок с провалами проб

Если статус Pod указывает на проблемы, связанные с пробами, вот как их можно устранить:

Проверьте события Pod

```
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 проб

- Тщательно проверьте параметры проб (путь, порт, команда, задержки, пороги) в вашем Deployment/Pod YAML. Частая ошибка — неверный порт или путь.

Проверьте код приложения

- Убедитесь, что эндпоинт проверки состояния реализован корректно и действительно отражает готовность/работоспособность 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 (`kubectl top pod <pod-name>`) и рассмотрите возможность настройки лимитов/запросов ресурсов.

Сетевые проблемы

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

Menu

Рабочие нагрузки

Deployments

Понимание Deployments

Создание Deployments

Управление Deployments

Устранение неполадок с помощью CLI

DaemonSets

Понимание DaemonSets

Создание DaemonSets

Управление DaemonSets

StatefulSets

Понимание StatefulSets

Создание StatefulSets

Управление StatefulSets

CronJobs

Понимание CronJobs

Создание CronJobs

Немедленное выполнение

Удаление CronJobs

Jobs

Понимание Jobs

Пример YAML файла

Обзор выполнения

Pods

Понимание Pod'ов

Пример YAML файла

Управление Pod с помощью CLI

Управление Pod с помощью веб-консоли

Контейнеры

Понимание контейнеров

Понимание эфемерных контейнеров

Взаимодействие с контейнерами

Deployments

Содержание

Понимание Deployments

Обратитесь к официальной документации Kubernetes: Deployments ↗

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

Deployment — это объект в API Kubernetes, который управляет Pods и ReplicaSets. При создании Deployment Kubernetes автоматически создает ReplicaSet, который отвечает за поддержание заданного 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, если они падают, завершаются или удаляются 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.

4. При обновлении Deployment (например, смена 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.

Создание Deployments

Создание Deployment с помощью CLI

Предварительные требования

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

Пример YAML файла

```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, управляемых этим Deployment
  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
```

Создание Deployment через YAML

```bash
# Шаг 1: Создать Deployment через 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, созданных этим Deployment
```

Создание Deployment 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 загрузить из него в текущем кластере.

Процедура — Настройка базовой информации

1. В **Container Platform** перейдите в **Workloads** > **Deployments** в левой боковой панели.
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 (docker-registry-projectname), включают имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

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

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

Параметры	Описание
More > **Update Strategy**	Настраивает стратегию `rollingUpdate` для обновлений без простоя: **Max surge** (`maxSurge`): • Максимальное количество Pod, которое может превышать 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 расчете 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 расчете от 10 реплик. **Примечания**: 1. **Значения по умолчанию**: `maxSurge=1` , `maxUnavailable=1` , если явно не заданы. 2. **Незапущенные Pod** (например, в состояниях `Pending` / `CrashLoopBackOff`) считаются недоступными. 3. **Одновременные ограничения**: • `maxSurge` и `maxUnavailable` не могут оба быть `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:

Параметры	Описание
	• `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 развертывании.

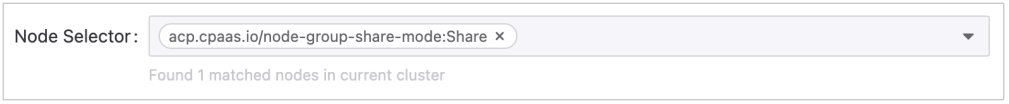
Процедура — Настройка Pod

Примечание: В кластерах с разной архитектурой, при развертывании образов одной архитектуры, убедитесь, что настроены правильные 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 смотрите Volume Mounting Guide.
Pull Secret	Требуется **только** при загрузке образов из сторонних реестров (через ручной ввод URL образа). **Примечание**: Секрет для аутентификации при загрузке образа из защищенного реестра.
Close Grace Period	Время (по умолчанию: `30s`), отведенное Pod для корректного завершения после получения сигнала на остановку. - В этот период Pod завершает текущие запросы и освобождает ресурсы. - Установка `0` приводит к немедленному удалению (SIGKILL), что может вызвать прерывание запросов.

1. **Node Affinity Rules**

Параметры	Описание
More > **Node Selector**	Ограничивает Pod узлами с определенными метками (например, `kubernetes.io/os: linux`). 
More > **Affinity**	Определяет тонкие правила планирования на основе существующих. **Типы Affinity**: • **Pod Affinity**: Планировать новые Pod на узлах, где уже размещены определённые Pod (одинаковый топологический домен). • **Pod Anti-affinity**: Запретить совместное размещение новых Pod с определёнными Pod. **Режимы применения**: • `requiredDuringSchedulingIgnoredDuringExecution`: Pod планируются *только* если правила выполнены. • `preferredDuringSchedulingIgnoredDuringExecution`: Приоритет узлам, удовлетворяющим правилам, но допускаются исключения. **Поля конфигурации**: • `topologyKey`: Метка узла, определяющая топологические домены (по умолчанию: `kubernetes.io/hostname`). • `labelSelector`: Фильтр целевых Pod с помощью запросов по меткам.

3. **Настройка сети**

- Kube-OVN

Параметры	Описание
Bandwidth Limits	Обеспечивает 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 подсети. Если не указано, используется подсеть по умолчанию для namespace.
Static IP Address	Привязка постоянных IP-адресов к Pod: • Несколько Pod из разных Deployments 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 жестким ограничением уникальности: • Каждый IP может быть привязан **только к одному Pod** в кластере. • **Критично**: Количество статических IP должно быть ≥ количеству реплик Pod.

Процедура — 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 смотрите Resource Units. **Коэффициент overcommit namespace**: • **Без коэффициента overcommit**: Если существуют квоты ресурсов namespace: Запросы/лимиты контейнера наследуют значения по умолчанию namespace (можно изменить). Если квот нет: Нет значений по умолчанию; задается пользовательский Request. • **С коэффициентом overcommit**: Запросы рассчитываются автоматически как `Limits / Overcommit ratio` (неизменяемо). **Ограничения**: • Request ≤ Limit ≤ максимальная квота namespace. • Изменение коэффициента overcommit требует пересоздания pod для вступления в силу. • Коэффициент overcommit отключает ручную настройку request. • При отсутствии квот namespace — нет ограничений ресурсов контейнера.
Extended Resources	Настройка расширенных ресурсов, доступных в кластере (например, vGPU, pGPU).
Volume Mounts	Конфигурация персистентного хранилища. Смотрите Инструкции по монтированию томов. **Операции**:

Параметры	Описание
	• Существующие тома pod: нажмите **Add** • Отсутствуют тома pod: 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) Смотрите 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**: Вывод `$MESSAGE`: `/bin/sh -c "while true; do echo $(MESSAGE); sleep 10; done"` Смотрите Определение команд↗.
More > **Environment Variables**	• Статические значения: прямые пары ключ-значение • Динамические значения: ссылки на ключи ConfigMap/Secret, поля pod (`fieldRef`), метрики

Параметры	Описание
	ресурсов (`resourceFieldRef`) **Примечание**: Переменные окружения переопределяют настройки образа/конфигурационного файла.
More > **Referenced ConfigMaps**	Внедрение целого 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) Смотрите Параметры проверки здоровья.
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: обеспечить монтирование родительского каталога (например, `c:/a` для `c:/a/b/c/*.log`)
More > **Exclude Log Files**	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**.

   Смотрите Init Containers↗. Init Container:

   1.1. Запускается перед контейнерами приложения (последовательное выполнение).

   1.2. Освобождает ресурсы после завершения.

   1.3. Удаление разрешено, если:

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

3. Нажмите **Create**.

Справочная информация

Инструкции 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 вызовут ошибку создания pod.
ConfigMap	Монтирует полные/частичные данные ConfigMap как файлы: • Полный ConfigMap: создает файлы с именами ключей под путем монтирования • Выбор subpath: монтирует конкретный ключ (например, `my.cnf`)

Тип	Назначение
Secret	Монтирует полные/частичные данные Secret как файлы: • Полный Secret: создает файлы с именами ключей под путем монтирования • Выбор subpath: 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. Смотрите Ephemeral Volumes
Empty Directory	Временное хранилище для совместного использования между контейнерами в одном pod: • Создается на узле при старте pod • Удаляется при удалении pod **Сценарий использования**: обмен файлами между контейнерами, временное хранение данных. Смотрите EmptyDir
Host Path	Монтирует директорию хоста (должна начинаться с `/`, например, `/volumepath`).

Проверки здоровья

- Пример YAML файла для проверок здоровья
- Параметры настройки проверок здоровья в веб-консоли

Управление Deployments

Управление Deployment с помощью CLI

Просмотр Deployment

- Проверьте, что Deployment создан.

```
kubectl get deployments
```

- Получите подробную информацию о вашем Deployment.

```
kubectl describe deployments
```

Обновление Deployment

Выполните следующие шаги для обновления Deployment:

1. Обновим Pods nginx, чтобы использовать образ nginx:1 .16.1.

```
kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1
```

или используйте следующую 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. Для просмотра статуса развертывания выполните:

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

Выполните `kubectl get rs`, чтобы увидеть, что Deployment обновил Pods, создав новый ReplicaSet и масштабируя его до 3 реплик, а старый ReplicaSet масштабируется до 0 реплик.

```
kubectl get rs
```

Выполнение `kubectl get pods` теперь должно показывать только новые Pods:

```
kubectl get pods
```

Масштабирование Deployment

Вы можете масштабировать Deployment с помощью следующей команды:

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

Откат Deployment

- Предположим, что вы ошиблись при обновлении Deployment, указав имя образа как `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
```

Удаление Deployment

Удаление Deployment также удалит управляемый им ReplicaSet и все связанные Pods.

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

Управление Deployment через веб-консоль

Просмотр 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.

Обновление 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.

Удаление Deployment

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. В выпадающем меню **Actions** нажмите кнопку **Delete** в колонке операций и подтвердите.

Устранение неполадок с помощью CLI

Если у Deployment возникают проблемы, вот несколько распространенных методов диагностики.

Проверка статуса Deployment

```
kubectl get deployment nginx-deployment
kubectl describe deployment nginx-deployment # Просмотр подробных событий и статуса
```

Проверка статуса ReplicaSet

```
kubectl get rs -l app=nginx
kubectl describe rs <replicaset-name>
```

Проверка статуса Pod

```
kubectl get pods -l app=nginx
kubectl describe pod <pod-name>
```

Просмотр логов

```
kubectl logs <pod-name> -c <container-name> # Просмотр логов конкретного контейнера
kubectl logs <pod-name> --previous          # Просмотр логов предыдущего
завершенного контейнера
```

Вход в Pod для отладки

```
kubectl exec -it <pod-name> -- /bin/bash # Вход в shell контейнера
```

Проверка конфигурации здоровья

Убедитесь, что livenessProbe и readinessProbe настроены корректно, а конечные точки проверки здоровья вашего приложения отвечают правильно. Устранение неполадок с проверками

Проверка лимитов ресурсов

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 один экземпляр **Pod** DaemonSet, независимо от их меток ролей, при условии, что они планируемы.
2. **Основные ограничения** (исключённые узлы)
   - Узлы, явно помеченные как `Unschedulable: true` (например, через `kubectl cordon`).
   - Узлы со статусом `NotReady`.

- Узлы с несовместимыми **Taints**, если 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
  namespace: kube-system
  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 нужны, чтобы DaemonSet мог запускаться на узлах control plane, удалите их, если ваши 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 Pod:
kubectl get daemonset fluentd-elasticsearch # Просмотр DaemonSet
kubectl get pods -l name=fluentd-elasticsearch -o wide # Проверка Pod, управляемых этим DaemonSet, на конкретных узлах
```

Создание DaemonSet через веб-консоль

Предварительные требования

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

Процедура — настройка базовой информации

1. В **Container Platform** перейдите в **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, вы можете фильтровать образы по **Already Integrated**. **Integration Project Name**, например, образы (docker-registry-projectname), где projectname — имя проекта 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 обновлений 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 и 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, которые соответствуют:
  - Критериям `nodeSelector` (если определены).
  - Конфигурации `tolerations` (позволяющей планировать Pod на узлах с taints).
  - Узел находится в состоянии `Ready` и `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

- Для получения списка всех DaemonSet в namespace:

```
kubectl get daemonsets -n <namespace>
```

- Для получения подробной информации 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, который хотите обновить.

3. В выпадающем меню **Actions** выберите **Update**, чтобы открыть страницу редактирования DaemonSet, где можно обновить `Replicas`, `image`, `updateStrategy` и др.

Удаление DaemonSet

1. В **Container Platform** перейдите в **Workloads** > **DaemonSets**.

2. Найдите DaemonSet, который хотите удалить.

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, обеспечивая:

- **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 обновления с прокруткой**: 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 идентичность Pod-ов
  replicas: 3 # задаёт желаемое количество реплик Pod-ов (по умолчанию: 1)
  minReadySeconds: 10 # по умолчанию 0
  template: # задаёт 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: # задаёт шаблоны PersistentVolumeClaim (PVC). Каждый Pod получает уникальный PersistentVolume (PV), динамически созданный на основе 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 вашего StatefulSet и связанных с ним 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 загрузить из него в текущем кластере.

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 (docker-registry-projectname), где projectname — имя проекта 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 рабочей нагрузки и ожидаемого объёма запросов.
Update Strategy	Управляет поэтапными обновлениями при прокрутке StatefulSet. Стратегия `RollingUpdate` является значением по умолчанию и рекомендуется. **Partition**: порог по порядковому номеру для обновления Pod-ов. • Pod-ы с индексом ≥ `partition` обновляются сразу. • Pod-ы с индексом < `partition` сохраняют предыдущую спецификацию. **Пример**: • `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 создавать постоянное хранилище для каждого Pod-а. Каждая реплика Pod-а в StatefulSet автоматически

Параметры	Описание

	получает свой уникальный PersistentVolumeClaim (PVC) на основе заранее определённых 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**: поддерживаются 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 хранения через storageClassName. Если не указано, используется StorageClass по умолчанию в кластере. Поддерживаются различные типы хранения в облаке и on-prem (например, SSD, HDD).
	• 4. **Ёмкость**: настраивается через resources.requests.storage. **Пример**: 1Gi. Поддерживается динамическое расширение томов, если это разрешено StorageClass.

Процедура — Настройка Pod

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 проверки состояния
- Параметры настройки проверки состояния 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 подробную информацию о вашем StatefulSet.

```
kubectl describe statefulsets
```

Масштабирование StatefulSet

- Чтобы изменить количество реплик для существующего StatefulSet:

```
kubectl scale statefulset <statefulset-name> --replicas=<new-replica-count>
```

- 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 обновление с прокруткой (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 обновления с прокруткой:

```
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) во избежание потери данных. Чтобы удалить также 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 политикой `reclaimPolicy` равной `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 т.д.

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 ↗
- Running Automated Tasks with a CronJob ↗

CronJob определяет задачи, которые выполняются до завершения и затем останавливаются. Они позволяют запускать одну и ту же Job несколько раз согласно расписанию.

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 копирования, очистки или рассылки 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 CronJobs через YAML

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

Создание CronJobs через веб-консоль

Предварительные требования

Получите адрес образа. Образы могут быть взяты из реестра образов, интегрированного администратором платформы через toolchain, либо из сторонних реестров образов.

- Для образов из интегрированного реестра администратор обычно назначает реестр образов вашему проекту, что позволяет использовать образы из 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.

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 (docker-registry-projectname) означает, что projectname — имя проекта платформы, а containers — имя проекта в реестре образов.
4. В разделе **Cron Configuration** настройте метод 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 учётом 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 учётом часового пояса (по умолчанию — локальный часовой пояс пользователя).

Параметр	Описание
Concurrency Policy	Определяет, как обрабатываются параллельные выполнения Job (`Allow` , `Forbid` или `Replace` согласно спецификации K8s ↗).

Job History Retention:

- Установите лимиты хранения для завершённых Jobs:
  - **History Limits**: лимит истории успешных задач (по умолчанию: 20)
  - **Failed Jobs**: лимит истории неудачных задач** (по умолчанию: 20)
- При превышении лимитов хранения старейшие задачи удаляются в первую очередь.

5. В разделе **Job Configuration** выберите тип задачи. CronJob управляет Jobs, состоящими из Pods. 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

Поиск ресурса 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. Нажмите **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 ещё не запланирован к выполнению.

Статус	Описание
Running	Pod(ы) Job активно выполняются.
Succeeded	Все Pods, связанные с Job, успешно завершились (код выхода 0).
Failed	По крайней мере один Pod, связанный с Job, завершился с ошибкой (код выхода не 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 завершений
  parallelism: 1 # Максимальное количество параллельных Pod
  backoffLimit: 3 # Максимальное количество попыток повторного запуска
  template:
    spec:
      restartPolicy: Never # Политика перезапуска для Job (Never/OnFailure)
      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:

- **Создать задание с помощью**

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

- **Отслеживать жизненный цикл задания 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 сайта Kubernetes: 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, Docker-контейнеров) с общими ресурсами хранения

и сети, а также спецификацией того, как запускать эти контейнеры. **Pods** являются фундаментальными строительными блоками, на которых основаны все контроллеры более высокого уровня (например, **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 container.
        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'ами 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 более 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, 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

Для проброса локального порта на порт Pod, что позволяет напрямую получить 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 с помощью веб-консоли

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 другие вкладки.

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 важны для планировщика Kubernetes, чтобы эффективно размещать Pod на узлах, а также для kubelet, чтобы обеспечивать управление ресурсами.

Параметр	Описание
	• **Requests**: минимально гарантированное количество CPU/памяти, необходимое для **запуска** и работы контейнера. Это значение используется планировщиком Kubernetes для выбора **Node**, на котором может работать Pod. • **Limits**: максимальное количество CPU/памяти, которое контейнер может потреблять во время выполнения. Превышение лимита CPU приводит к ограничению (throttling), а превышение лимита памяти — к завершению контейнера с ошибкой Out Of Memory (OOM Killed). Для подробного описания единиц измерения (например, `m` для milliCPU, `Mi` для мебибайт) см. Resource Units. **Логика расчёта ресурсов на уровне Pod** Эффективные значения Requests и Limits CPU и памяти для Pod вычисляются как сумма и максимум значений отдельных контейнеров. Метод расчёта для Requests и Limits на уровне Pod аналогичен; в этом документе приведена логика на примере Limit. Если Pod содержит только стандартные контейнеры (рабочие контейнеры): Эффективное значение Limit CPU/памяти Pod — это сумма Limit CPU/памяти всех контейнеров внутри Pod. **Пример**: если Pod содержит два контейнера с Limit CPU/памяти 100m/100Mi и 50m/200Mi соответственно, то агрегированный Limit CPU/памяти Pod будет 150m/300Mi. Если Pod содержит как initContainers, так и стандартные контейнеры: Шаги расчёта Limit CPU/памяти Pod следующие: • 1. Определить максимальное значение Limit CPU/памяти среди всех initContainers. • 2. Вычислить сумму Limit CPU/памяти всех стандартных контейнеров. • 3. Сравнить результаты шагов 1 и 2. Комплексный Limit CPU/памяти Pod будет максимальным из значений CPU (максимум из initContainers и сумма стандартных контейнеров) и

Параметр	Описание
	максимальным из значений памяти (максимум из initContainers и сумма стандартных контейнеров). **Пример расчёта**: если Pod содержит два initContainers с Limit CPU/памяти 100m/200Mi и 200m/100Mi, максимальный эффективный Limit для initContainers будет 200m/200Mi. Одновременно, если Pod содержит два стандартных контейнера с Limit CPU/памяти 100m/100Mi и 50m/200Mi, общая сумма Limit для стандартных контейнеров будет 150m/300Mi. Таким образом, комплексный Limit CPU/памяти Pod будет max(200m, 150m) для CPU и max(200Mi, 300Mi) для памяти, что равно 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 запланирован и работает Pod.
Service Account	Service Account — это объект Kubernetes, предоставляющий 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 находится в состоянии, влияющем на бизнес-процессы, например `Pending` или `CrashLoopBackOff`, ручное удаление Pod после устранения ошибки поможет ему быстро вернуться в желаемое состояние, например `Running`. При этом удалённые Pod будут пересозданы на текущем узле или переназначены.

- 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 со статусом `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 действие.

4. (Массовое удаление) Выберите Pod для удаления, нажмите **Delete** над списком и 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.

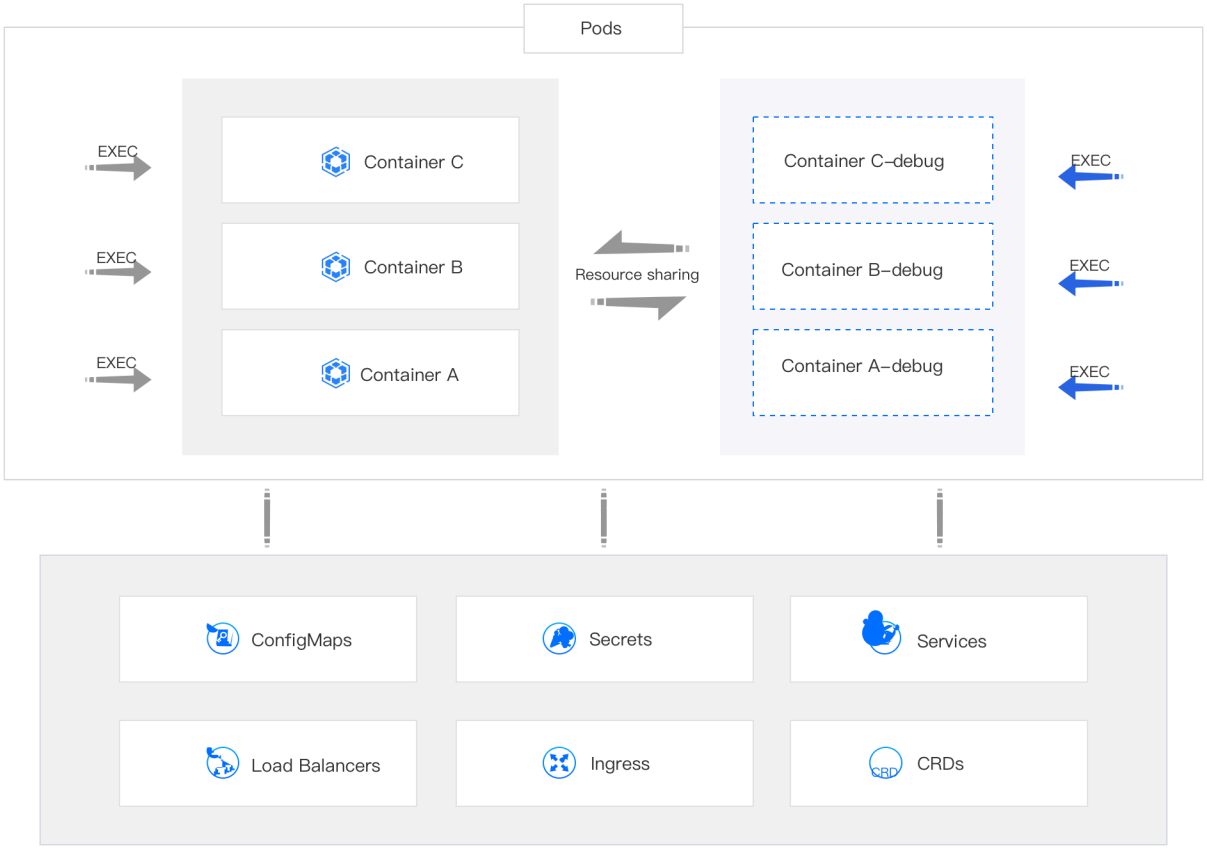
Хотя часто можно выполнять команды непосредственно внутри запущенного контейнера с помощью kubectl exec, многие производственные образы контейнеров намеренно минимальны и могут не содержать необходимых утилит для отладки (например, bash, net-tools, tcpdump), чтобы уменьшить размер образа и поверхность атаки. Эфемерные контейнеры решают эту проблему, предоставляя предварительно настроенную 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 и логов Pod, часто требуется более глубокая интерактивная отладка непосредственно в среде выполнения Pod.
- **Настройка конфигурации и эксперименты**: если текущая конфигурация приложения демонстрирует неоптимальное поведение, можно 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 **эфемерных контейнеров**. 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` в том же Pod.



:::Notes * Нельзя добавить эфемерный контейнер, напрямую изменяя статический манифест Pod (PodSpec). Функция эфемерных контейнеров предназначена для динамического внедрения в запущенные Pods, обычно через API-вызовы (например, kubectl debug). * **Эфемерные контейнеры**, 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 находится 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 - 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, присоединённые к Pod:

  ```
  kubectl describe pod my-nginx-pod
  ```

  Найдите 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 запросит **инициализацию** (например, для настройки необходимой 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. Подсказка: нажмите опцию «Command Query» в правом верхнем углу терминала, чтобы увидеть список распространённых инструментов отладки и примеры их использования.

> **INFO**
>
> Нажмите «Command Query» в правом верхнем углу, чтобы просмотреть распространённые инструменты и их использование.

6. По завершении 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 возможности для загрузки и скачивания файлов из контейнеров и в контейнеры.

Взаимодействие с контейнерами через CLI

Exec

Для выполнения команды внутри конкретного контейнера в Pod (полезно для получения shell, запуска диагностических команд и т.д.):

```
kubectl exec -it <pod-name> -c <container-name> -n <namespace> -- <command>
# -it: Обеспечивает интерактивный режим и TTY (псевдо-терминал) для сессии shell.
# -c: Указывает имя целевого контейнера в Pod. Опускается, если контейнер один.
# --: 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-path> -c
<container-name>
# -c: (Опционально) Указывает целевой контейнер, если в Pod несколько.

# 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: Скачивание 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. Также 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** для входа в окно веб-консоли и выполнения командной строки.
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** для входа в окно веб-консоли и выполнения командной строки.

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. Ключевые 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 основе Helm и представляет собой комплексную платформу управления распространением Chart, расширяющую ограничения CLI-инструмента Helm. 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 релизами, тогда как в Alauda Container Platform используются ресурсы HelmRequest для определения, развертывания и управления Helm charts. Основные отличия:

1. **Декларативный vs Императивный**: HelmRequest обеспечивает декларативный подход к развертыванию Helm, в то время как традиционный Helm CLI — императивный.
2. **Нативность Kubernetes**: HelmRequest — это кастомный ресурс, напрямую интегрированный с API Kubernetes.
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, такими как ресурсы 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 релизами (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): Используется 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: `docker pull nginx`
2. Сохраните в tar-пакет: `docker save nginx > nginx.latest.tar`
3. Загрузите и отправьте в приватный реестр:

```
docker load -i nginx.latest.tar
docker tag nginx:latest 192.168.80.8:30050/nginx:latest
docker 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/$NAMESPACE/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\":\"IfNotPresent\"}}"
    },
    "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/$NAMESPACE/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/$NAMESPACE/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-
system/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 условия

Репозитории шаблонов добавляются администраторами платформы. Пожалуйста, обратитесь к администратору платформы, чтобы получить имена 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	Если выбран тип 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 проблему с файлами в загруженном архиве, например, отсутствие содержимого или неправильное 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.

Menu

Конфигурации

Настройка ConfigMap

Понимание ConfigMap

Ограничения ConfigMap

Пример ConfigMap

Создание ConfigMap через веб-консоль

Создание ConfigMap с помощью CLI

Операции

Просмотр, редактирование и удаление через CLI

Способы использования ConfigMap в Pod

ConfigMap и Secret

Настройка Secrets

Понимание Secrets

Создание Secret типа Opaque

Создание Secret типа Docker registry

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 описываются ConfigMap и способы их создания и использования.

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 применяться для хранения конфигурационных данных системных компонентов, таких как контроллеры. Например:

```
# my-app-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
  namespace: default
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 применяться 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 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. Рекомендуется рассматривать каждый компонент, настроенный с помощью ConfigMap, индивидуально.
- 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. Не поддерживаются поды, созданные с помощью флагов `--manifest-url`, `--config` узла OpenShift Container Platform или его REST API, так как 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
  namespace: k-1
data:
  APP_ENV: "production"
  LOG_LEVEL: "debug"
```

Создание ConfigMap через веб-консоль

1. Перейдите в **Container Platform**.
2. В левой боковой панели нажмите **Configuration** > **ConfigMap**.
3. Нажмите **Create ConfigMap**.
4. Следуйте инструкциям ниже для настройки соответствующих параметров.

Параметр	Описание
Entries	Относится к парам `ключ:значение`, поддерживает методы **Add** и **Import**. • **Add**: Вы можете добавлять элементы конфигурации по одному или вставить одну или несколько строк с парами key=value в поле ввода Key для массового добавления элементов конфигурации. • **Import**: Импортируйте текстовый файл размером не более 1M. Имя файла будет использоваться как ключ, а содержимое файла — как значение для элемента конфигурации.
Binary Entries	Относится к бинарным файлам размером не более 1M. Имя файла будет использоваться как ключ, а содержимое файла — как значение для элемента конфигурации.

Параметр	Описание
	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 -n k-1
```

Операции

Вы можете нажать (⋮) справа на странице списка или нажать **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 рекомендуется заранее ознакомиться с инструкциями по

эксплуатации.

Операции	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 новая конфигурация вступила в силу. • 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 -n k-1 -o yaml
kubectl edit configmap app-config -n k-1
kubectl delete configmap app-config -n k-1
```

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

```
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

```
env:
  - name: APP_ENV
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: APP_ENV
```

ConfigMap и Secret

Особенность	ConfigMap	Secret
Тип данных	Не содержит чувствительной информации	Чувствительные данные (например, пароли)
Кодирование	Обычный текст	Кодируется в Base64
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 цель — предотвратить прямое внедрение чувствительных данных в определения Pod или образы контейнеров, что повышает безопасность и переносимость.

Secrets похожи на ConfigMaps, но предназначены специально для конфиденциальных данных. Обычно они хранятся в base64-кодировке 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 конфиденциальную информацию 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 случаев использования. Платформа обычно 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 протокола TLS (Transport Layer Security), часто используемых для HTTPS и безопасного ingress.
- **SSH Key**: используется для хранения приватных SSH-ключей, часто 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 (kubernetes.io/dockerconfigjson)**: хранит JSON-строку аутентификации, необходимую для загрузки образов контейнеров из приватных репозиториев образов (Docker Registry).

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 смонтированных файлов (тома)**: Secrets могут монтироваться в виде файлов в том пода, позволяя приложениям читать конфиденциальные данные по заданному пути.

Примечание: Экземпляры Pod в рабочих нагрузках 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 типа Docker registry

```
kubectl create secret docker-registry my-docker-creds \
  --docker-username=myuser \
  --docker-password=mypass \
  --docker-server=https://index.docker.io/v1/ \
  --docker-email=my@example.com
```

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: my-docker-creds
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOnsidXNlcm5hbWUiOiJteXVzZXIiLCJwYXNzd29yZ
```

K8s автоматически преобразует ваше имя пользователя, пароль, email и информацию о сервере в стандартный формат входа Docker:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "username": "myuser",
      "password": "mypass",
      "email": "my@example.com",
      "auth": "bXl1c2VyOm15cGFzcw=="  # base64(username:password)
    }
  }
}
```

Этот JSON затем кодируется в base64 и используется в поле data Secret.

Используйте его в Pod:

```
imagePullSecrets:
  - name: my-docker-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, вебхуками 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 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 смонтированных файлов (тома)

```
volumes:
  - name: secret-volume
    secret:
      secretName: my-secret

volumeMounts:
  - name: secret-volume
    mountPath: "/etc/secret"
```

Последующие действия

При создании рабочих нагрузок 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, чтобы при необходимости обновить или удалить Secret.

Операция	Описание
Обновить	После добавления или обновления Secret, рабочие нагрузки, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, должны пересоздать свои Pods, чтобы новые настройки вступили в силу.
Удалить	• После удаления Secret рабочие нагрузки, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, могут столкнуться с проблемами из-за отсутствия источника ссылки при пересоздании Pods. • Пожалуйста, не удаляйте Secrets, автоматически созданные платформой, так как это может нарушить работу платформы.

Операция	Описание
	Например: Secrets типа service-account-token, содержащие информацию аутентификации для ресурсов namespace, и Secrets в системных namespace (таких как kube-system).

Menu

Наблюдаемость приложения

Мониторинговые панели

Предварительные требования

Панели мониторинга на уровне Namespace

Мониторинг на уровне рабочих нагрузок

Логи

Процедура

События

Процедура

Интерпретация записей событий

Мониторинговые панели

- Поддерживается просмотр данных мониторинга ресурсов для компонентов рабочих нагрузок на платформе за последние 7 дней (с возможностью настройки периода хранения данных мониторинга). Включает статистику по приложениям, рабочим нагрузкам, подам, а также тенденциям/рейтингу использования CPU/памяти.
- Поддерживается мониторинг на уровне Namespace.
- Поддерживаемый мониторинг на уровне рабочих нагрузок: **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** создаёт 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 рабочих нагрузок

> В этом разделе 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 стандартных метрик мониторинга.

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** отражает фактическое потребление ресурсов.
Memory	**Utilization/Usage**	**Utilization** = Usage/Limit (лимиты) Метод оценки аналогичен CPU. Высокий показатель может привести к нестабильности 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), получаемые/отправляемые подами.
Disk Rate	**Read/Write**	Скорость чтения/записи (байт/сек) смонтированных томов на рабочую нагрузку.
Disk IOPS	**Read/Write**	Операции ввода/вывода в секунду (IOPS) смонтированных томов на рабочую нагрузку.

Пользовательская панель мониторинга

1. Нажмите **Toggle Icon** для переключения на пользовательские панели. Руководствуйтесь инструкцией Добавление панели в пользовательскую панель для создания выделенной панели мониторинга на уровне **Workload**.

INFO

Наведите курсор на кривые графика, чтобы просмотреть метрики по каждому поду и выражения PromQL в конкретные 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.
Previous Logs	Просмотр логов завершённых контейнеров (доступно, если container restartCount > 0).

Операция	Описание
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 для просмотра полного содержимого в модальном окне.

WARNING

- **Обработка усечения**: Записи логов, превышающие 2000 символов, будут усечены с добавлением многоточия `...`
  - Усечённые части не могут быть найдены функцией поиска на странице.
  - Нажмите на маркер многоточия `...` в усечённых строках для просмотра полного содержимого 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 настройкам ротации логов Kubernetes. Для исторического анализа используйте Logs в разделе Observe.

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 контейнерного рантайма 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 соответствующие 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 All Events.

Menu

Как сделать

Настройка правил срабатывания планировщика задач

Правила срабатывания планировщика 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 1:42 AM по UTC.

Пример 2: пекинское время 4:32 AM 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 8:32 PM предыдущего дня по UTC.

Запись выражений Crontab

Базовый формат и диапазон значений Crontab: `minute hour day month weekday`, с соответствующими диапазонами значений, приведёнными в таблице ниже:

Минуты	Часы	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.
- `*`: обозначает весь период времени. Например, для минут — каждую минуту.
- `/`: используется для указания шага увеличения значений. Например: `n/m` означает начиная с n, увеличивать на m каждый раз.

Ссылка на инструмент преобразования ↗

Распространённые примеры:

- Ввод `30 18 25 12 *` означает, что задача сработает в `18:30:00 25 декабря`.
- Ввод `30 18 25 * 6` означает, что задача сработает в `18:30:00 каждую субботу`.
- Ввод `30 18 * * 6` означает, что задача сработает в `18:30:00 по субботам`.
- Ввод `* 18 * * *` означает, что задача сработает каждую минуту, начиная с `18:00:00` (включая `18:00:00`).
- Ввод `0 18 1,10,22 * *` означает, что задача сработает в `18:00:00` 1-го, 10-го и 22-го числа каждого месяца.
- Ввод `0,30 18-23 * * *` означает, что задача сработает в 00 и 30 минут каждого часа с 18:00 до 23:00 ежедневно.
- Ввод `* */1 * * *` означает, что задача сработает каждую минуту.

- Ввод `* 2-7/1 * * *` означает, что задача сработает каждую минуту с 2 AM до 7 AM ежедневно.
- Ввод `0 11 4 * mon-wed` означает, что задача сработает в `11:00 AM` 4-го числа каждого месяца и по понедельникам, вторникам и средам.

Menu

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 для контейнеров

Включение метаданных в образы

Управление образами

Обзор политики загрузки образа

Разрешение pod ссылаться на образы из других защищенных реестров

Создание pull secret

Использование pull secret в рабочей нагрузке

Обзор образов

Содержание

Понимание контейнеров и образов

Контейнеры и образы — важные понятия, которые необходимо понимать при создании и управлении контейнированным 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 или Docker. Образ — это бинарный файл, который включает все необходимые

компоненты для запуска одного контейнера, а также метаданные, описывающие его требования и возможности.

Можно рассматривать образ как технологию упаковки. Контейнеры имеют доступ только к ресурсам, определённым в образе, если им 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 `docker` 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 своим хешем — длинным шестнадцатеричным числом, например fd44297e2ddb050ec4f…, которое обычно сокращается до 12 символов, например fd44297e2ddb.

Реестр образов

Реестр образов — это сервер контента, который может хранить и предоставлять образы контейнеров. Например:

- Docker Hub ↗
- 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, их идентифицирующих. Например, образы Jenkins для Alauda Container Platform находятся в репозитории:

```
docker.io/alauda/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:

```
docker.io/alauda/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:

```
docker.io/alauda/jenkins-2-centos7@sha256:ab312bda324
```

Контейнеры

Основными единицами приложений в Alauda Container Platform являются контейнеры. Технологии Linux-контейнеров — 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 прочего друг друга. Обычно каждый контейнер предоставляет одну службу, часто называемую микросервисом, например веб-сервер или базу данных, хотя контейнеры могут использоваться для любых рабочих нагрузок.

Ядро Linux уже много лет включает возможности для технологий контейнеров. Проект Docker разработал удобный интерфейс управления Linux-контейнерами на хосте. В последнее время Open Container Initiative↗ разработала открытые стандарты для форматов контейнеров и сред выполнения контейнеров. Alauda Container Platform и Kubernetes добавляют возможность оркестрации контейнеров форматов OCI и Docker в многокластерных установках.

Хотя при использовании Alauda Container Platform вы не взаимодействуете напрямую со средами выполнения контейнеров, понимание их возможностей и терминологии важно для понимания их роли в Alauda Container Platform и работы ваших приложений внутри контейнеров.

Menu

Как сделать

Создание образов

Изучение лучших практик для контейнеров

Включение метаданных в образы

Управление образами

Обзор политики загрузки образа

Разрешение pod ссылаться на образы из других защищенных реестров

Создание pull secret

Использование pull secret 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 для контейнеров

При создании контейнерных образов для запуска на Alauda Container Platform автору образа следует учитывать ряд лучших практик, чтобы обеспечить хороший опыт для потребителей этих образов. Поскольку образы предназначены быть неизменяемыми и использоваться как есть, следующие рекомендации помогают сделать ваши образы максимально удобными и простыми в использовании 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 соответствующем upstream-образе, используя инструкцию FROM. Это гарантирует, что ваш образ сможет легко получить исправления безопасности из upstream-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 latest-версии upstream-образа.

Поддерживайте совместимость внутри тегов

При тегировании собственных образов старайтесь сохранять обратную совместимость внутри одного тега. Например, если вы предоставляете образ с именем image и в настоящее время он включает версию `1.0`, вы можете использовать тег `image:v1`. При обновлении образа, если он остается совместимым с оригинальным, вы можете продолжать использовать тег `image:v1`, и потребители этого тега смогут получать обновления без сбоев.

Если позже вы выпустите несовместимое обновление, переключитесь на новый тег, например `image:v2`. Это позволит потребителям самостоятельно перейти на новую версию, не ломая их работу 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 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 коммуникации. 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 подготовки перед запуском основного процесса. Если ваш образ использует такой скрипт, он должен использовать `exec`, чтобы процесс скрипта был заменён вашим программным обеспечением. Если не использовать `exec`, сигналы, посылаемые средой выполнения контейнера, 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 `docker run -i`, который запускает скрипт, а тот — процесс. Если вы хотите остановить контейнер с помощью `CTRL+C`, и если скрипт использует `exec` для запуска сервера, `docker` отправит SIGINT серверному процессу, и всё сработает как ожидается. Если `exec` не используется, SIGINT будет отправлен процессу скрипта, а серверный процесс продолжит работу.

Также учтите, что ваш процесс работает как `PID 1` в контейнере. Это означает, что если основной процесс завершится, весь контейнер остановится, отменяя все дочерние процессы, запущенные из `PID 1`.

Очистка временных файлов

Удаляйте все временные файлы, созданные в процессе 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, создав команду `RUN` следующим образом:

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clean 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, которые не удалятся при последующем `yum clean`. Эти файлы не видны в конечном 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 значит, что если вы выполните команду `rm` в более позднем слое, хотя файлы и будут скрыты, общий размер образа для загрузки не уменьшится. 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 одном `RUN` уменьшает количество слоёв в образе, что улучшает время загрузки и распаковки.

Размещайте 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 редко меняющиеся инструкции в начале `Dockerfile`. Это гарантирует, что последующие сборки того же образа будут очень быстрыми, так как кэш не будет инвалидирован изменениями в верхних слоях.

Например, если вы работаете с `Dockerfile`, который содержит команду `ADD` для установки файла, над которым вы работаете, и команду `RUN` для установки пакета через `yum`, лучше поместить `ADD` последним:

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

Так при каждом изменении `myfile` и повторной сборке `docker build` система повторно использует кэш для команды `yum` и создает 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 команда `ADD` инвалидирует кэш слоя `RUN`, и операция `yum` будет выполнена заново.

Отмечайте важные порты

Инструкция `EXPOSE` делает порт в контейнере доступным для хост-системы и других контейнеров. Хотя можно указать порт для экспонирования при запуске с помощью `docker run`, использование `EXPOSE` в `Dockerfile` облегчает использование вашего образа как людьми, так и программным обеспечением, явно объявляя порты, необходимые для работы вашего ПО:

- Экспонированные порты отображаются в `docker ps` для контейнеров, созданных из вашего образа.
- Экспонированные порты присутствуют в метаданных образа, возвращаемых `docker inspect`.
- Экспонированные 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 определение версии без просмотра `Dockerfile`. Другой пример — объявление 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 в продакшене пользователь получит известный пароль. Вместо этого пароли должны настраиваться 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 задать.

Избегайте sshd

Лучше не запускать `sshd` в вашем образе. Вы можете использовать команду `docker exec` для доступа к контейнерам, запущенным на локальном хосте. Также можно

использовать `docker exec` для доступа к контейнерам, запущенным в кластере Alauda Container Platform. Установка и запуск `sshd` в образе открывает дополнительные векторы атак и требует регулярного обновления безопасности.

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 монтируется заново. 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 уничтожения контейнера, должны записываться в том. Движки контейнеров поддерживают флаг `readonly` для контейнеров, который можно использовать для строгого соблюдения правил не записывать данные во временное хранилище контейнера. Проектирование образа с учётом этой возможности сейчас облегчит её использование в будущем.

Явное определение томов в вашем `Dockerfile` помогает потребителям образа понять, какие тома необходимо определить при запуске вашего образа.

См. документацию Kubernetes ↗ для получения дополнительной информации о том, как тома используются в Alauda Container Platform.

> **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 использования 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.

См. документацию Docker ↗ для получения дополнительной информации об инструкции `LABEL`.

Имена меток обычно имеют пространство имён. Пространство имён устанавливается в соответствии с проектом, который будет использовать метки. Для Kubernetes пространство имён — io.k8s.

См. документацию Docker по пользовательским метаданным ↗ для деталей формата.

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 ваши сборки и развертывания работали.

Политика загрузки образа

Каждый контейнер в pod использует образ контейнера. После того как вы создали образ и загрузили его в реестр, вы можете ссылаться на него в pod.

Содержание

Обзор политики загрузки образа

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 контейнера не указан, Alauda Container Platform устанавливает его в зависимости от тега образа:

1. Если тег latest, по умолчанию устанавливается Always.
2. В противном случае по умолчанию устанавливается 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 других проектов Alauda Container Platform или из защищенных реестров, необходима дополнительная настройка.

Разрешение pod ссылаться на образы из других защищенных реестров

Чтобы загрузить защищенный контейнер из других приватных или защищенных реестров, необходимо создать pull secret на основе учетных данных вашего контейнерного клиента, например `Docker`, и добавить его в вашу учетную запись сервиса.

Docker использует конфигурационный файл для хранения данных аутентификации для входа в защищенный или незащищенный реестр:

По умолчанию Docker использует файл $HOME/.docker/config.json.

Эти файлы хранят информацию для аутентификации, если вы ранее входили в защищенный или незащищенный реестр.

Создание pull secret

Вы можете получить секрет для загрузки образа из приватного реестра контейнерных образов или репозитория. Подробнее см. Pull an Image from a Private Registry ↗.

Использование pull secret в рабочей нагрузке

Вы можете использовать pull secret, чтобы разрешить рабочим нагрузкам загружать образы из приватного реестра одним из следующих способов:

- Связав секрет с `ServiceAccount`, что автоматически применит секрет ко всем pod, использующим эту учетную запись сервиса.
- Определив `imagePullSecrets` в спецификации pod, что удобно для таких сред, как GitOps или ArgoCD.

Вы можете использовать секрет для загрузки образов pod, добавив секрет в вашу учетную запись сервиса. Обратите внимание, что имя учетной записи сервиса должно совпадать с именем учетной записи, которую использует pod.

Пример вывода:

```
apiVersion: v1
imagePullSecrets:
- name: default-dockercfg-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 GitOps-процессов, 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 workflow ArgoCD:

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: <example_workflow>
spec:
  entrypoint: <main_task>
  imagePullSecrets:
  - name: <pull_secret_name>
```

Menu

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 Registry через YAML

Обновление/Удаление Alauda Container Platform Registry

Установка через Web UI

Когда использовать этот метод?

Предварительные требования

Установка плагина кластера Alauda Container Platform Registry через веб-консоль

Обновление/Удаление Alauda Container Platform Registry

Руководство пользователя

Распространённые операции с командами CLI

Вход в реестр

Добавление разрешений на пространство имён для пользователей

Добавление разрешений на пространство имён для сервисного аккаунта

Загрузка образов (Pulling Images)

Отправка образов (Pushing Images)

Using Alauda Container Platform Registry in Kubernetes Clusters

Registry Access Guidelines

Deploy Sample Application

Cross-Namespace Access

Best Practices

Verification Checklist

Troubleshooting

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 повышенную 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 надежность данных образов.

Одним из основных концептов дизайна является логическая изоляция и управление на основе Namespace. В 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 явно не предоставлено разрешение.

Аутентификация и авторизация

Механизм аутентификации и авторизации Alauda Container Platform Registry глубоко интегрирован с системой аутентификации и авторизации на уровне платформы ACP, обеспечивая контроль доступа с детализацией до уровня namespace:

Аутентификация

Пользователям или автоматизированным процессам (например, CI/CD пайплайнам на платформе, автоматическим задачам сборки и т.д.) не нужно поддерживать отдельный набор учетных данных для Registry. Аутентификация происходит через стандартные механизмы платформы (например, с использованием предоставляемых платформой API токенов, интегрированных корпоративных систем идентификации и т.п.). При доступе к Alauda Container Platform Registry через CLI или другие инструменты обычно используется существующая сессия входа в 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 прав пользователя или ServiceAccount в соответствующем namespace.

- **Как правило**, владельцу или разработчику namespace автоматически предоставляются права Push и Pull для репозиториев образов в этом namespace.

- **Пользователи из других namespace или пользователи, желающие выполнять pull образов между namespace**, должны получить соответствующие права от администратора целевого namespace (например, привязать роль, позволяющую выполнять pull образов через RBAC) перед тем, как получить доступ к образам в этом namespace.
- **Этот 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

- **Легковесное 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 полного рабочего процесса через интегрированные решения Source to Image (S2I) при недостаточном использовании инфраструктуры.

Menu

Установка

Установка через YAML

Когда использовать этот метод?

Предварительные требования

Установка Alauda Container Platform Registry через YAML

Обновление/Удаление Alauda Container Platform Registry

Установка через Web UI

Когда использовать этот метод?

Предварительные требования

Установка плагина кластера Alauda Container Platform Registry через веб-консоль

Обновление/Удаление Alauda Container Platform Registry

Установка через YAML

Содержание

Когда использовать этот метод?

Рекомендуется для:

- **Продвинутых пользователей** с опытом работы с Kubernetes, предпочитающих ручной подход.
- **Промышленных 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.
- **Иметь 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: internal-docker-registry
  labels:
    create-by: cluster-transformer
    manage-delete-by: cluster-transformer
    manage-update-by: cluster-transformer
  name: internal-docker-registry
spec:
  config:
    access:
      address: ""
      enabled: false
    fake:
      replicas: 2
    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/SecretName
      ingressClassName: "<INGRESS-CLASS-NAME>"  # [REQUIRED] IngressClassName
      insecure: false
    persistence:
      accessMode: ReadWriteMany
      nodes: ""
      path: <YOUR-HOSTPATH>  # [REQUIRED] Локальный путь для LocalVolume
      size: <STORAGE-SIZE>  # [REQUIRED] Размер хранилища (например, 10Gi)
      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
самоподписанных сертификатов
      region: <S3-REGION>                        # Регион S3
      regionEndpoint: <S3-ENDPOINT>  # Конечная точка S3
      secretName: <S3-CREDENTIALS-SECRET>            # Секрет с учетными данными
S3
    service:
      nodePort: ""
      type: ClusterIP
  pluginName: internal-docker-registry
```

2. **Настройте следующие поля** в соответствии с вашей средой:

```
spec:
  config:
    global:
      oidc:
        ldapID: "<LDAP-ID>"                  # LDAP ID
    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-storage
    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_ID/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 вашем кластере:

```
kubectl apply -f registry-plugin.yaml
```

Справочник по конфигурации

Обязательные поля

Параметр	Описание	Пример значения
`spec.config.global.oidc.ldapID`	LDAP ID для аутентификации OIDC	`ldap-test`
`spec.config.ingress.hosts[0].name`	Пользовательский домен для доступа к registry	`registry.yourcompany.com`
`spec.config.ingress.hosts[0].tlsCert`	Ссылка на секрет TLS сертификата (namespace/secret-name)	`cpaas-system/registry-tls`
`spec.config.ingress.ingressClassName`	Имя класса ingress для registry	`cluster-alb-1`

Параметр	Описание	Пример значения
`spec.config.persistence.size`	Размер хранилища для registry	`10Gi`
`spec.config.persistence.storageClass`	Имя StorageClass для registry	`nfs-storage-sc`
`spec.config.s3storage.bucket`	Имя S3 bucket для хранения образов	`prod-image-store`
`spec.config.s3storage.region`	Регион AWS для S3 хранилища	`us-west-1`
`spec.config.s3storage.regionEndpoint`	URL конечной точки сервиса S3	`https://s3.amazonaws.com`
`spec.config.s3storage.secretName`	Секрет с учетными данными S3	`s3-access-keys`

Проверка

1. Проверьте плагин:

```
kubectl get clusterplugininstances internal-docker-registry -o yaml
```

2. Проверьте поды registry:

```
kubectl get pods -n cpaas-system -l app=internal-docker-registry
```

Обновление/Удаление Alauda Container Platform Registry

Обновление

Выполните следующую команду на 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=<CLUSTER-
NAME>,cpaas.io/module-name=internal-docker-registry -o name)
```

Удаление

Выполните следующую команду на 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=internal-docker-registry -o name | xargs kubectl delete -n
cpaas-system
```

Установка через Web UI

Содержание

Когда использовать этот метод?

Рекомендуется для:

- **Пользователей, делающих первые шаги**, предпочитающих пошаговый визуальный интерфейс.
- **Быстрых прототипов** в непроизводственных средах.
- Команд с **ограниченными знаниями Kubernetes**, которым нужен упрощённый процесс развертывания.
- Сценариев с **минимальной кастомизацией** (например, стандартные настройки хранилища).
- **Базовых сетевых настроек** без специфических правил ingress.
- Конфигураций **StorageClass** для обеспечения высокой доступности.

Не рекомендуется для:

- Производственных сред, требующих продвинутых настроек хранения (S3 storage).

- Сетевых конфигураций с необходимостью специфических 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** для доступа к списку плагинов кластера.
3. Найдите плагин кластера **Alauda Container Platform Registry**, нажмите **Install**, затем перейдите на страницу установки.
4. Настройте параметры согласно приведённым ниже 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 несёт значительные риски безопасности и должно включаться с крайней осторожностью.
Enable IPv6	Включите эту опцию, если кластер использует одноадресную 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 Storage Type = LocalVolume)
StorageClass	Выберите StorageClass. При количестве реплик более 1 выберите хранилище с поддержкой RWX (ReadWriteMany) (например, File Storage) для обеспечения высокой доступности. (Доступно только при Storage Type = 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 ресурсов 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 Docker клиент 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.

Menu

Руководство пользователя

Распространённые операции с командами CLI

Вход в реестр

Добавление разрешений на пространство имён для пользователей

Добавление разрешений на пространство имён для сервисного аккаунта

Загрузка образов (Pulling Images)

Отправка образов (Pushing Images)

Using Alauda Container Platform Registry in Kubernetes Clusters

Registry Access Guidelines

Deploy Sample Application

Cross-Namespace Access

Best Practices

Verification Checklist

Troubleshooting

Распространённые операции с командами CLI

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 сейчас работаете — my-ns.

> Свяжитесь с технической службой, чтобы получить плагин kubectl-acp и убедитесь, что он правильно установлен в вашей среде.

Содержание

Вход в реестр

Войдите в реестр кластера, выполнив вход в ACP.

```
kubectl acp login <ACP-endpoint>
```

Добавление разрешений на пространство имён для пользователей

Добавить пользователю разрешение на pull в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-puller --user=
<username> -n <namespace>
```

Добавить пользователю разрешение на push в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-pusher --user=
<username> -n <namespace>
```

Добавление разрешений на пространство имён для сервисного аккаунта

Добавить сервисному аккаунту разрешение на pull в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-puller --
serviceaccount=<namespace>:<serviceaccount-name> -n <namespace>
```

Добавить сервисному аккаунту разрешение на push в пространстве имён.

```
kubectl create rolebinding <binding-name> --clusterrole=system:image-pusher --
serviceaccount=<namespace>:<serviceaccount-name> -n <namespace>
```

Загрузка образов (Pulling Images)

Загружает образ из реестра внутрь кластера (например, для развертывания Pod).

```
# Загрузить образ с именем my-app и тегом latest из реестра текущего пространства имён (my-ns)
kubectl acp pull registry.cluster.local/my-ns/my-app:latest

# Загрузить образы из других пространств имён (например, shared-ns) (требуется разрешение на pull из пространства shared-ns)
kubectl acp pull registry.cluster.local/shared-ns/base-image:latest
```

Эта команда проверяет вашу личность и разрешения на pull в целевом пространстве имён, а затем загружает образ из реестра.

Отправка образов (Pushing Images)

Отправляет локально собранные образы или образы, загруженные из других источников, в определённое пространство имён реестра.

Сначала необходимо с помощью стандартного инструмента командной строки для контейнеров, например docker, присвоить локальному образу тег с адресом и форматом пространства имён целевого реестра.

```
# Присвоить тег с целевым адресом:
docker tag my-app:latest registry.cluster.local/my-ns/my-app:v1

# Использовать команду kubectl для отправки в реестр текущего пространства 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.io/demo/my-app:latest
# Используйте команду kubectl для отправки его в пространство имён (my-ns) реестра
kubectl acp push remote.registry.io/demo/my-app:latest registry.cluster.local/my-ns/my-
app:latest
```

Эта команда проверяет вашу личность и разрешения на push в пространстве имён my-ns, а затем загружает локально промаркированный 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

Registry Access Guidelines

- **Рекомендуется использовать внутренний адрес**: Для образов, хранящихся в реестре кластера, всегда отдавайте предпочтение внутреннему сервисному адресу `internal-docker-registry.cpaas-system.svc` при развертывании внутри кластера. Это обеспечивает оптимальную сетевую производительность и избегает ненужной внешней маршрутизации.
- **Использование внешнего адреса**: Внешний ingress-домен (например, `registry.cluster.local`) предназначен в первую очередь для:

- загрузки/выгрузки образов из вне кластера (например, с машин разработчиков, систем CI/CD)
- операций вне кластера, требующих доступа к реестру

Deploy Sample Application

1. Создайте приложение с именем `my-app` в пространстве имён `my-ns`.
2. Сохраните образ приложения в реестре по адресу `internal-docker-registry.cpaas-system.svc/my-ns/my-app:v1`.
3. **По умолчанию** ServiceAccount в каждом пространстве имён автоматически настраивается с imagePullSecret для доступа к образам из `internal-docker-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: internal-docker-registry.cpaas-system.svc/my-ns/my-app:v1
        ports:
        - containerPort: 8080
```

Cross-Namespace Access

Чтобы разрешить пользователям из `my-ns` вытягивать образы из `shared-ns`, администратор `shared-ns` может создать role binding для предоставления необходимых прав.

Example Role Binding

```
# Доступ к образам из другого пространства имён (требуются права)
kubectl create rolebinding cross-ns-pull \
  --clusterrole=system:image-puller \
  --serviceaccount=my-ns:default \
  -n shared-ns
```

Best Practices

- **Использование реестра**: Всегда используйте `internal-docker-registry.cpaas-system.svc` для развертываний, чтобы обеспечить безопасность и производительность.
- **Изоляция пространств имён**: Используйте изоляцию пространств имён для лучшей безопасности и управления образами.
  - Используйте пути образов, основанные на пространстве имён: `internal-docker-registry.cpaas-system.svc/<namespace>/<image>:<tag>`.
- **Контроль доступа**: Используйте role bindings для управления доступом между пространствами имён для пользователей и сервисных аккаунтов.

Verification Checklist

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>
```

Troubleshooting

- **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 bindings в целевом пространстве имён.
- **Проблемы с сетью**: Проверьте сетевые политики и конфигурации сервисов, чтобы обеспечить подключение к внутреннему реестру.
- **Сбои DNS**: Проверьте содержимое файла `/etc/hosts` на узле, убедитесь, что разрешение DNS для `internal-docker-registry.cpaas-system.svc` настроено правильно.
  - Проверьте конфигурацию /etc/hosts узла для корректного разрешения DNS `internal-docker-registry.cpaas-system.svc`
  - Пример отображения сервиса реестра (ClusterIP сервиса internal-docker-registry):

```
# /etc/hosts
127.0.0.1   localhost localhost.localdomain
10.4.216.11 internal-docker-registry.cpaas-system internal-docker-registry.cpaas-
system.svc internal-docker-registry.cpaas-system.svc.cluster.local # cpaas-
generated-node-resolver
```

  - **Как получить текущий ClusterIP** `internal-docker-registry`:

```
kubectl get svc -n cpaas-system internal-docker-registry -o
jsonpath='{.spec.clusterIP}'
```

Menu

Source to Image

Обзор

Введение

Концепция Source to Image

Основные возможности

Основные преимущества

Сценарии применения

Ограничения использования

Архитектура

Примечания к выпуску

Примечания к выпуску Alauda Container Platform Builds

Поддерживаемые версии

Примечания к выпуску v1.1

Политика жизненного цикла

Установка

Installing Alauda Container Platform Builds

Prerequisites

Procedure

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

Menu

Обзор

Введение

Концепция Source to Image

Основные возможности

Основные преимущества

Сценарии применения

Ограничения использования

Архитектура

Примечания к выпуску

Примечания к выпуску Alauda Container Platform Builds

Поддерживаемые версии

Примечания к выпуску v1.1

Политика жизненного цикла

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 технологиям, 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, что обеспечивает 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 сосредоточиться на написании 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.
- **Управление полным 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**: поддержка пользовательских плагинов и расширений для удовлетворения специфических требований.

Основные преимущества

- **Ускорение разработки**: оптимизирует процесс сборки, ускоряя доставку приложений.
- **Повышенная гибкость**: поддержка сборки на нескольких языках программирования.
- **Повышенная эффективность**: автоматизация процессов сборки и развертывания, сокращая ручное вмешательство.
- **Повышенная надежность**: предоставляет подробные логи сборки и визуальный мониторинг для упрощения 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 собирать и развертывать веб-приложения, просто указав URL репозитория с кодом.

- 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

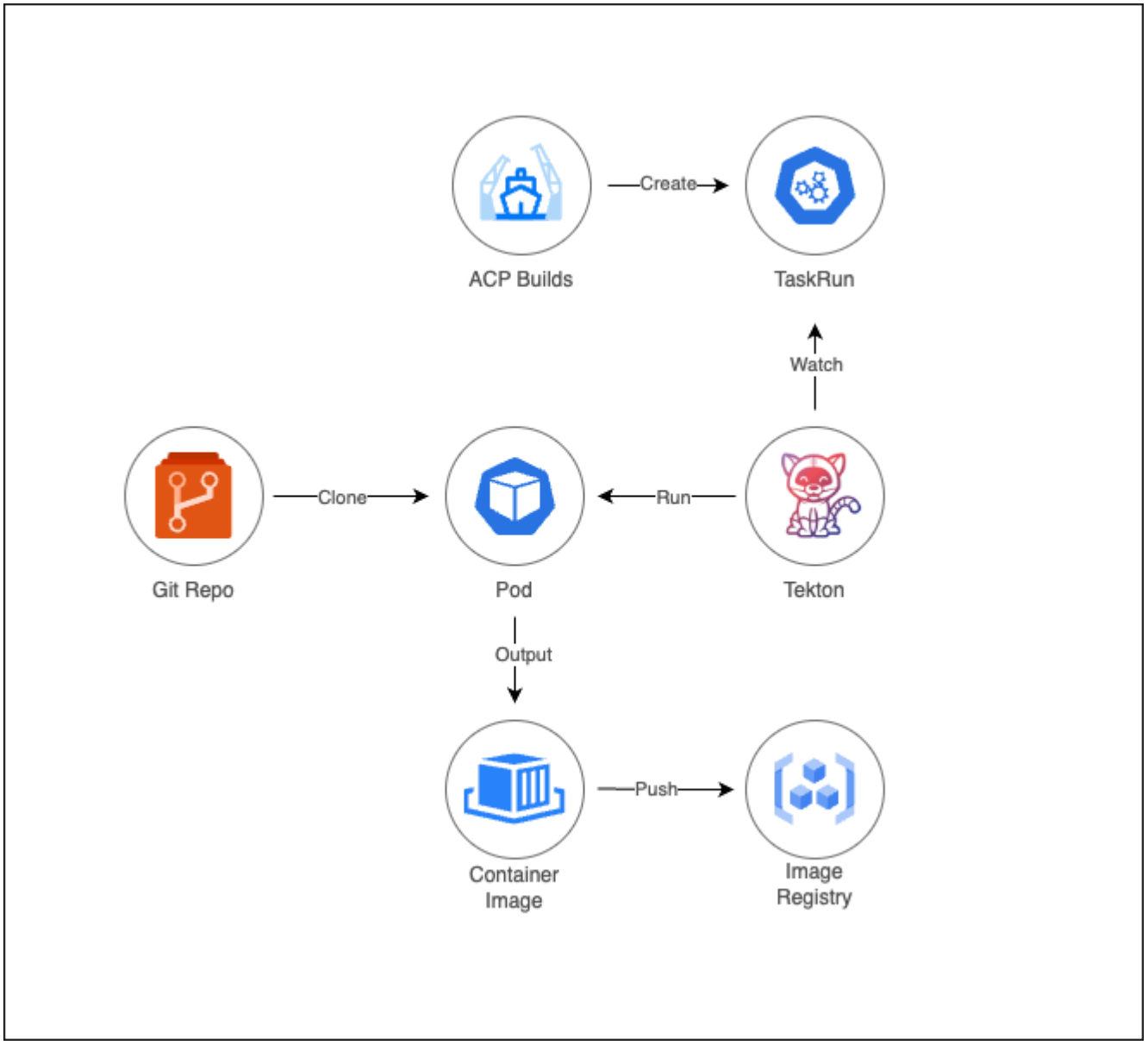
> Текущая версия поддерживает только языки Java, Go, Python и Node.js.

WARNING

Требования: Alauda DevOps Pipelines operator ↗ теперь доступен в кластере OperatorHub.

Menu

Архитектура



Возможность Source to Image (S2I) реализована через оператор **Alauda Container Platform Builds**, обеспечивающий автоматическую 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
v1.1.0	**v4.1**	**v4.1**

Примечания к выпуску v1.1

v1.1.0

1. Устранение уязвимостей безопасности.
2. Независимый выпуск.

Политика жизненного цикла

График жизненного цикла версии

Ниже приведён график жизненного цикла выпущенных версий оператора Alauda Container Platform Builds:

Version	Release Date	End of Life
v1.1.0	2025-08-15	2027-08-15

Menu

Установка

Installing Alauda Container Platform Builds

Содержание

Prerequisites

> Alauda Container Platform Builds — это контейнерный инструмент, предлагаемый Alauda Container Platform , который объединяет сборку (поддерживает 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.

Procedure

Install the Alauda Container Platform Builds Operator

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:

Parameter	Recommended Configuration
Channel	`Alpha`: По умолчанию выбран канал **alpha**.
Version	Выберите последнюю версию.
Installation Mode	`Cluster`: Один оператор используется во всех пространствах имён кластера для создания и управления экземплярами, что снижает потребление ресурсов.
Namespace	`Recommended`: Рекомендуется использовать пространство имён **shipyard-operator**; оно будет создано автоматически, если отсутствует.
Upgrade Strategy	Выберите `Manual`. • `Manual`: При появлении новой версии в OperatorHub • действие **Upgrade** не будет выполняться автоматически.

1. На странице **Install** выберите конфигурацию по умолчанию, нажмите **Install** и завершите установку оператора **Alauda Container Platform Builds**.

Install the Shipyard instance

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 экземпляра можно использовать конфигурацию по умолчанию, если нет специальных требований.

5. Нажмите **Create**.

Verification

- После успешного создания экземпляра 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 инструкции Creating an application from Code.

Menu

Обновление

Обновление 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 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` доступны 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**↗.

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.

Menu

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 проблем, анализ и устранение неполадок.

Предварительные требования

- Завершена установка Installing Alauda Container Platform Builds.
- Требуется доступ к репозиторию образов; если он отсутствует, 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, который интегрирован с платформой и уже выделен для текущего проекта; платформа поддерживает GitLab, GitHub и Bitbucket. • **Input**: Используйте URL репозитория кода, который не интегрирован 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 с последним коммитом 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 репозитория кода типа Input можно при необходимости добавить секрет аутентификации.

	Builder Image	• Образ, включающий конкретные среды выполнения языков программирования, библиотеки зависимостей и скрипты S2I. Его основная задача — преобразовывать исходный код 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). Подробности см. в: Installing Alauda Container Platform Builds 2. Создайте ресурсы типа Build на основе вышеуказанных стратегий и информации, введённой в форме. Эти ресурсы определяют стратегии сборки, параметры сборки, репозитории исходного кода, репозитории выходных образов и другую связанную 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 через конвейер Tekton и создаёт Pod для выполнения процесса сборки. Pod отвечает за фактическую работу по сборке, которая включает: получение 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 запросы. При включённом внешнем доступе весь соответствующий трафик будет перенаправлен на этот порт для предоставления внешних сервисов. • **Другие параметры**: Пожалуйста, ознакомьтесь с описанием 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), а также их YAML.
Start Build	При сбое сборки или изменении исходного кода можно нажать эту кнопку для повторного запуска задачи сборки.

Menu

Как сделать

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 этой 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**.

Menu

Стратегия изоляции узлов

Стратегия изоляции узлов предоставляет стратегию изоляции узлов на уровне проекта, которая позволяет проектам использовать узлы кластера эксклюзивно.

Введение

Введение

Преимущества

Сценарии применения

Архитектура

Архитектура

Основные понятия

Основные понятия

Изоляция узлов

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

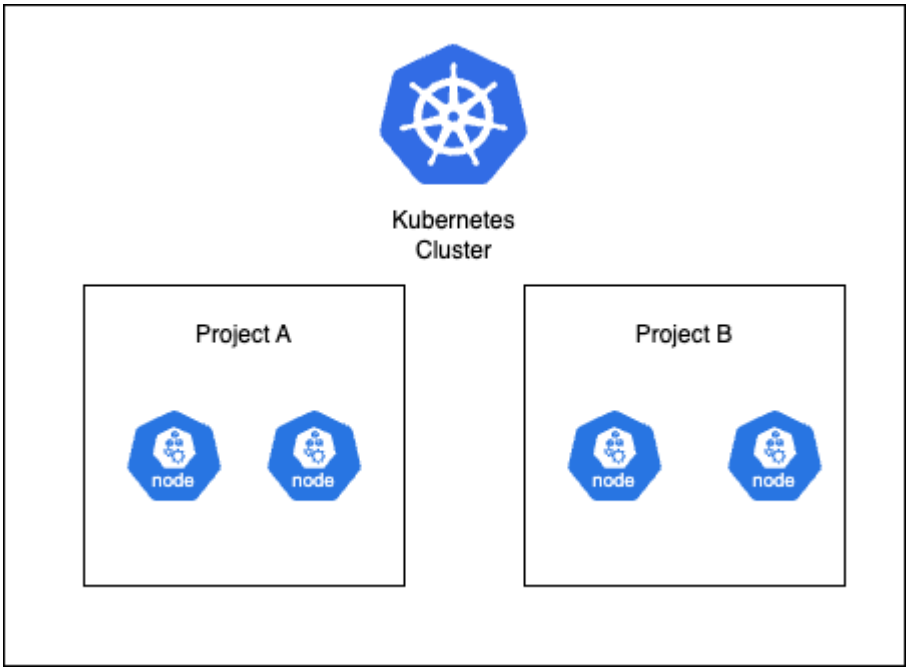
Удобное выделение узлов проектам в эксклюзивном или совместном режиме, предотвращая конкуренцию за ресурсы между проектами.

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 по производительности.

Menu

Архитектура



Node Isolation Strategy реализована на основе компонента Container Platform Cluster Core, обеспечивая возможность изоляции узлов между проектами за счёт выделения узлов в каждом кластере нагрузки. При создании контейнеров в проекте они принудительно планируются на узлы, выделенные именно этому проекту.

Menu

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 конкуренции за ресурсы и ухудшения производительности.

Menu

Руководства

Создание стратегии изоляции узлов

Создание стратегии изоляции узлов

Удаление 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 Pods 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 Pods в указанном проекте из политики могут запускаться на узлах, включённых в политику; при выключенном переключателе Pods из других проектов

Параметр	Описание
	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 настроена только одна политика изоляции узлов; если проект уже имеет назначенную политику 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 Pods в проекте, не соответствующие текущей политике, будут запланированы на узлы, включённые в текущую политику, после их пересоздания;
- При включённом параметре **Project Exclusivity** Pods, 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**.

Menu

Разрешения

Функция	Действие	Platform Administrator	Platform auditors	Project Manager	Namespac Administrat
nodegroups `acp-nodegroups`	Просмотр	✓	✓	✓	✓
	Создать	✓	×	×	×
	Обновить	✓	×	×	×
	Удалить	✓	×	×	×

Часто задаваемые вопросы

Содержание

Почему при импорте namespace не должно быть нескольких ResourceQuota?

При импорте namespace, если в нем содержится несколько ресурсов ResourceQuota, платформа выберет минимальное 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 — это минимальное значение между `a` и `b`.

Поскольку Kubernetes проверяет каждый ResourceQuota независимо, при наличии нескольких ResourceQuota в namespace, после импорта рекомендуется удалить все ResourceQuota, кроме `default`. Это помогает избежать сложностей в расчетах квот из-за нескольких ResourceQuota, что может привести к ошибкам.

Почему при импорте namespace не должно быть нескольких LimitRanges?

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 — 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 быть только один LimitRange. После импорта рекомендуется удалить все LimitRange, кроме того, который называется `default`, чтобы избежать непредсказуемого поведения из-за нескольких LimitRange.