Menu

Разработчик

Обзор

Введение

Преимущества

Сценарии использования

Сквозные принципы облачно-нативных технологий

Основные понятия

Особенности

Создание приложения

Управление namespace

Наблюдаемость приложения

Source to Image

Registry

Стратегия изоляции узлов

OAM Application

Быстрый старт

Создание простого приложения через образ

Введение

Важные замечания

Предварительные требования

Обзор рабочего процесса

Процедура

Создание приложений

Обзор

Управление пространствами имён

Управление жизненным циклом приложений

Управление рабочими нагрузками Kubernetes

Основные понятия

Руководства

Как сделать

Реестр

Введение

Принципы и изоляция namespace

Аутентификация и авторизация

Преимущества

Сценарии применения

Установка

Руководство пользователя

Source to Image

Введение

Концепция Source to Image

Основные возможности

Основные преимущества

Сценарии применения

Ограничения использования

Установка

Архитектура

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 LimitRange или LimitRange с именем, отличным от `default`?

Menu

Обзор

Введение

Введение

Преимущества

Сценарии использования

Сквозные принципы облачно-нативных технологий

Основные понятия

Описание единиц ресурсов

Типы приложений

Типы рабочих нагрузок

Особенности

Особенности

Создание приложения

Управление namespace

Наблюдаемость приложения

Source to Image

Registry

Стратегия изоляции узлов

OAM Application

Введение

Модуль Developer view предоставляет разработчикам возможности оркестрации и эксплуатации облачно-нативных приложений. Он обеспечивает единый интерфейс для компоновки приложений из различных источников с интеграцией встроенных инструментов наблюдаемости для эксплуатации в продакшене.

Содержание

Преимущества

> Модуль Developer view обеспечивает следующие ключевые преимущества:

1. Унифицированная оркестрация приложений

   - Images: Развертывание из публичных/приватных реестров с использованием образов
   - YAML: Прямое объявление ресурсов Kubernetes с проверкой схемы
   - Source to Image (S2I): Сборка контейнеризованных приложений непосредственно из исходного кода
   - Helm Charts: Развертывание упакованных приложений из курируемого Application Catalog

- Реализует компоновку приложений в соответствии с GitOps с использованием нескольких подходов

2. Комплексное управление жизненным циклом

> Реализует декларативное управление нагрузками и namespace:

- Progressive Delivery: Canary/Blue-Green развертывания через ServiceMesh
- Управление ресурсами:
  - Provisioning namespace с политиками RBAC
  - Политики распределения ресурсов через HPA/VPA
  - Динамическое масштабирование с интеграцией Cluster Autoscaler
- Автоматизация рабочих процессов: интеграция CI/CD pipeline с Tekton

1. Корпоративное управление namespace

> Реализует управление namespace с поддержкой мультиарендности:

- Полное управление жизненным циклом
- Гарантии ресурсов:
  - Конфигурации ResourceQuota и LimitRange
  - Настраиваемые коэффициенты overcommit для CPU/Memory

1. Полноценная наблюдаемость

> Интегрированный стек мониторинга с:

- Корреляция событий: интеграция Kubernetes Event и Audit логов
- Аналитика логов: агрегирование логов
- Метрики и дашборды: мониторинг и пользовательские правила оповещений

Сценарии использования

> Основные сценарии использования модуля Developer включают:

- Мультиоблачное развертывание

Организации распределяют нагрузки между несколькими облачными провайдерами (AWS, Azure, GCP), чтобы избежать зависимости от одного поставщика, оптимизировать затраты и обеспечить отказоустойчивость. Облачная доставка приложений обеспечивает единообразные пайплайны развертывания, абстрагирующие особенности конкретных провайдеров.

- Гибридные облачные среды

Предприятия поддерживают инфраструктуру на месте наряду с ресурсами публичного облака. Облачная доставка обеспечивает единое развертывание приложений в гибридных средах, управляя сложностями гетерогенной инфраструктуры.

- Интеграция edge computing

По мере роста популярности edge computing приложения должны работать в централизованных облаках, edge-устройствах и региональных edge-узлах. Облачная доставка расширяет возможности развертывания на эти распределённые edge-среды.

- Пайплайн от разработки до продакшена

Облачные методологии позволяют бесшовно продвигать приложения от разработки через тестирование/стейджинг к продакшену, сохраняя согласованность конфигураций и учитывая особенности каждой среды.

- Глобальные мульти-региональные развертывания

Для глобально распределённых приложений облачная доставка обеспечивает единообразные развертывания по географическим регионам, оптимизируя задержки и соблюдая требования локализации данных.

- Восстановление после сбоев и непрерывность нагрузки

Облачная доставка облегчает создание среды восстановления после сбоев, идентичной продакшену, обеспечивая быстрое переключение и непрерывность работы.

Сквозные принципы облачно-нативных технологий

> Эти сценарии опираются на основные облачно-нативные принципы:

- Контейнеризация
- Infrastructure-as-Code (IaC)
- Декларативные конфигурации
- Неизменяемая инфраструктура
- GitOps workflows

Они обеспечивают согласованность, надёжность и автоматизацию в гетерогенных вычислительных средах.

Menu

Основные понятия

Описание единиц ресурсов

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

Menu

Типы приложений

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

- **Application**: Полноценное бизнес-приложение, состоящее из одного или нескольких связанных вычислительных компонентов (Workloads), внутренних маршрутов (Services) и других нативных ресурсов Kubernetes. Поддерживает создание через редактирование в UI, YAML-оркестрацию и шаблоны, может работать в средах разработки, тестирования или производства. Различные типы нативных приложений можно создавать следующими способами:
  - **Create from Image**: Быстрое создание приложений с использованием существующих контейнерных образов.
  - **Create from Catalog**: Создание приложений с использованием пакетов Helm Chart.
  - **Create from YAML**: Создание приложений с использованием YAML-конфигурационных файлов.
  - **Create from Code**: Создание приложений с использованием исходного кода.
- **Operator Backed App**: На основе компонентов приложения (Operator backed) можно быстро развернуть компонентное приложение и использовать возможности Operators для автоматизации всего жизненного цикла управления приложением.
- **OAM Application**: Используется для определения модели облачно-нативных приложений. По сравнению с логикой оркестрации контейнеров или Kubernetes, OAM больше фокусируется на самом «приложении». На основе OAM общие возможности приложений инкапсулируются в высокоуровневые интерфейсы для использования на протяжении всего процесса развертывания, разработки и эксплуатации приложений.

Menu

Типы рабочих нагрузок

Помимо создания нативных приложений и компонентных приложений, рабочие нагрузки также можно создавать напрямую в **Container Platform > Computing Components**:

- **Deployment**: Наиболее часто используемый контроллер рабочих нагрузок для развертывания бессостоящих приложений. Он обеспечивает запуск заданного количества реплик Pod в кластере, поддерживает rolling update и откаты, подходит для сценариев бессостоящих приложений, таких как веб-сервисы и API-сервисы.
- **DaemonSet**: Обеспечивает запуск реплики Pod на каждом узле кластера (или на определённых узлах). При добавлении узла в кластер Pod автоматически создаётся; при удалении узла из кластера соответствующие Pods также удаляются. Подходит для сценариев, требующих запуска логирования, мониторинга и т.п. на каждом узле.
- **StatefulSet**: Контроллер рабочих нагрузок для управления состоянием приложений. Он сохраняет фиксированную идентичность для каждого Pod и обеспечивает стабильное хранилище и сетевую идентичность, которые не меняются даже при пересоздании Pod. Подходит для состояних приложений, таких как базы данных и распределённые кэши.
- **Job**: Рабочая нагрузка для выполнения одноразовых задач. Job создаёт один или несколько Pods и гарантирует, что заданное количество Pods успешно завершит задачу перед завершением. Подходит для пакетной обработки, миграции данных и других одноразовых задач.
- **CronJob**: Используется для управления Jobs, запланированными на выполнение по расписанию. Можно задать временное выражение для запуска задачи, и система автоматически создаст и выполнит Job в назначенное время. Подходит для периодических задач, таких как резервное копирование данных, генерация отчётов и периодическая очистка.

Помимо создания вышеуказанных вычислительных компонентов через форму платформы, платформа также поддерживает создание Pods и Containers с помощью CLI-инструментов:

- **Pod**: Наименьшая единица развертывания в Kubernetes, Pod может содержать один или несколько контейнеров, которые совместно используют хранилище, сеть и конфигурационные декларации. Pods обычно управляются контроллерами (например, Deployments).
- **Container**: Стандартная программная единица, которая упаковывает код и все зависимости, позволяя приложениям быстро и надёжно запускаться в различных вычислительных средах. Контейнеры работают внутри Pods и используют ресурсы Pod.

Особенности

Содержание

Создание приложения

- Создание приложения

  Поддержка нескольких способов создания приложения, включая образ, yaml, код и каталог.

- Операции с приложением

  Использование приложения для оркестрации и управления рабочими нагрузками и связанными с ними ресурсами.

- Управление рабочими нагрузками

  Управление жизненным циклом рабочих нагрузок.

Управление namespace

- Управление жизненным циклом namespace

  Управление жизненным циклом namespace.

- Управление квотами и лимитами ресурсов

  Управление квотами и лимитами ресурсов namespace.

- Overcommit ресурсов namespace

  Разрешение overcommit ресурсов namespace.

Наблюдаемость приложения

- Логи

  Запрос истории логов или логов в реальном времени приложений.

- События

  Запрос событий, собранных с приложений.

- Мониторинг

  Мониторинг состояния приложения и генерация оповещений при возникновении аномалий.

Source to Image

- Сборка образа из исходного кода

  Сборка образа из исходного кода git-репозитория и отправка образа в репозиторий образов.

Registry

- Встроенный сервер Registry

  Простое развертывание сервера registry, доступного для платформы.

Стратегия изоляции узлов

- Изоляция узлов

  Поддержка изоляции узлов на уровне проекта для предотвращения конкуренции ресурсов между проектами.

OAM Application

- Эффективная эксплуатация и сопровождение

  Через OAM-приложения специалисты по эксплуатации и сопровождению могут сосредоточиться на бизнес-логике и управлять приложениями с точки зрения приложения, а не платформы, снижая порог для эксплуатации и сопровождения приложений. Специалисты по эксплуатации платформы могут централизованно управлять плагинами платформы, плагинами эксплуатации и другими конфигурациями, что повышает эффективность эксплуатации.

- Портативность

  Модель OAM-приложения включает конфигурации, связанные с эксплуатацией и сопровождением приложений, управлением сервисами и т.д. По сравнению с приложениями, развернутыми через Operators, Charts и другими методами, OAM-приложения можно многократно развертывать через YAML, что облегчает миграцию между средами. Даже без Kubernetes и конкретных вендоров OAM-приложения могут нормально работать на различных платформах.

- Масштабируемость

  Несколько типов компонентов, предустановленных на платформе, могут удовлетворить большинство потребностей разработки приложений: сетевые сервисы, stateful-приложения и нативные ресурсы Kubernetes. Кроме того, платформа предоставляет возможность расширения компонентов и traits, что облегчает разработчикам использование кастомных и инкапсулированных компонентов и traits.

Menu

Быстрый старт

Создание простого приложения через образ

> Это техническое руководство демонстрирует, как эффективно создавать, управлять и получать доступ к контейнеризованным приложениям в Alauda Container Platform с использованием нативных для Kubernetes методов.

Содержание

Введение

Сценарии использования

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

Время выполнения

Оценочное время выполнения: 10-15 минут

Важные замечания

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

Предварительные требования

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

Обзор рабочего процесса

№	Операция	Описание
1	Создать Namespace	Установить границу изоляции ресурсов
2	Настроить репозиторий образов	Настроить источники контейнерных образов
3	Создать приложение через Deployment	Создать workload Deployment
4	Экспонировать сервис через NodePort	Настроить сервис NodePort
5	Проверить доступность приложения	Тестировать доступ к конечной точке

Процедура

Создание namespace

> Namespaces обеспечивают логическую изоляцию для группировки ресурсов и управления квотами.

Предварительные условия

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

Процесс создания

1. Войдите в систему и перейдите в **Project Management** > **Namespaces**

2. Выберите **Create Namespace**

3. Настройте основные параметры:

** Параметр **	**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

Настройка репозитория образов

Alauda Container Platform поддерживает несколько стратегий получения образов:

Метод 1: Встроенный реестр через Toolchain

1. Перейдите в **Platform Management** > **Toolchain** > **Integration**

2. Создайте новую интеграцию:

Параметр	**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

Метод 2: Внешние сервисы реестров

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

Требование к проверке

- Сеть кластера должна иметь исходящий доступ к конечным точкам реестра

Создание приложения через Deployment

> Deployment обеспечивает декларативное обновление реплик Pod.

Процесс создания

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

Операции управления

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

Экспонирование сервиса через NodePort

Сервисы обеспечивают сетевой доступ к группам Pod.

Процесс создания

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

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

3. Подтвердите создание.

Критично

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

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

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

Параметр	Описание
Name	Введите имя сервиса.
Type	`NodePort`
Workload Name	Выберите ранее созданный `Deployment`.
Port	**Service Port**: номер порта, который сервис открывает внутри кластера, т.е. Port, например, `8080`. **Container Port**: целевой номер порта (или имя), сопоставленный с портом сервиса, т.е. targetPort, например, `80`.

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

Проверка доступности приложения

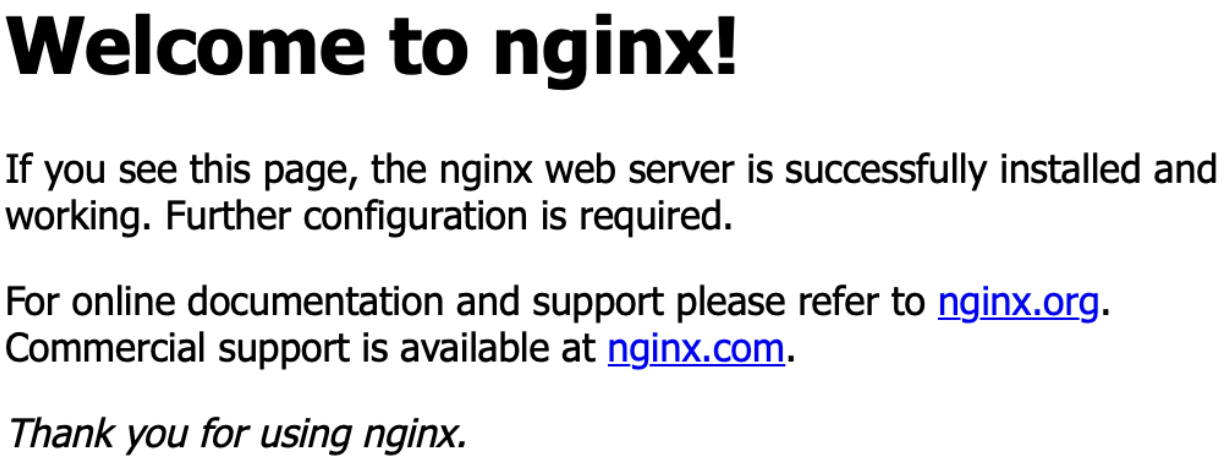
Метод проверки

1. Получите компоненты открытой конечной точки:

- **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

Обзор

Обзор

Управление пространствами имён

Управление жизненным циклом приложений

Управление рабочими нагрузками Kubernetes

Основные понятия

Понимание параметров

Обзор

Основные понятия

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Устранение распространённых проблем

Расширенные шаблоны использования

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Расширенные шаблоны использования

Понимание переменных окружения

Overview

Core Concepts

Use Cases and Scenarios

CLI Examples and Practical Usage

Best Practices

Руководства

Пространства имён

Подготовка перед созданием приложения

Создание приложений

Конфигурация после создания приложения

Эксплуатация и сопровождение

Наблюдаемость 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

Pod

Как сделать

Настройка правил срабатывания планировщика задач

Преобразование времени

Запись выражений Crontab

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 использованием 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 подов.
- 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 заранее 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 напрямую входящие в него 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 эксплуатацией платформа предоставляет логи, события, мониторинг и др.

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

Управление рабочими нагрузками Kubernetes

Поддержка основных типов рабочих нагрузок:

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

Menu

Основные понятия

Понимание параметров

Обзор

Основные понятия

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Устранение распространённых проблем

Расширенные шаблоны использования

Понимание команд запуска

Overview

Core Concepts

Сценарии использования

Примеры CLI и практическое использование

Лучшие практики

Расширенные шаблоны использования

Понимание переменных окружения

Overview

Core Concepts

Use Cases and Scenarios

CLI Examples and Practical Usage

Best Practices

Понимание параметров

Содержание

Обзор

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 приложения без необходимости пересборки образов.

Основные понятия

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, сохраняя 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 параметров конфигурации приложения:

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

2. Развёртывание для разных окружений

Разные параметры для различных 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, чтобы добавить 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**: включая справочный текст и примеры
- **Проверяйте ввод**: валидируйте значения параметров и выводите сообщения об ошибках

2. Вопросы безопасности

```
# ❌ Избегайте передачи конфиденциальных 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 для настройки контейнеризованных приложений в Kubernetes. Понимая, как правильно использовать параметры, вы можете создавать гибкие, переиспользуемые и удобные в сопровождении развёртывания, которые адаптируются к разным окружениям и требованиям.

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), формируя полную командную строку

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 args	ENTRYPOINT + CMD	`args: ["new-args"]`	ENTRYPOINT + new-args
Переопределение command	ENTRYPOINT + CMD	`command: ["new-cmd"]`	new-cmd
Переопределение обоих	ENTRYPOINT + CMD	`command: ["new-cmd"]` `args: ["new-args"]`	new-cmd + new-args

---

Сценарии использования

1. Пользовательский запуск приложения

Запуск разных приложений с использованием одного базового образа:

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

2. Отладка и устранение неполадок

Переопределение стандартной команды для запуска оболочки с целью отладки:

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

3. Скрипты инициализации

Запуск пользовательских скриптов 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 целей:

```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

Многошаговая 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 завершения 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, неинициализированных переменных и ошибках в пайпах

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

      log "Начинается инициализация приложения..."

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

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

3. Вопросы безопасности

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

4. Управление ресурсами

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

Расширенные шаблоны использования

1. Init-контейнеры с пользовательскими командами

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

2. Sidecar-контейнеры с разными командами

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

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

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

3. Шаблоны Job с пользовательскими командами

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

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

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 образов
- стандартный способ передачи конфигурационных данных приложениям
- доступные через стандартные API операционной системы на любом языке программирования

Источники переменных окружения в Kubernetes

Kubernetes поддерживает несколько источников переменных окружения:

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

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

Приоритет переменных окружения

Переменные окружения переопределяют конфигурацию в следующем порядке:

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

Use Cases and Scenarios

1. Конфигурация приложения

Основные 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 окружений

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 конкретного окружения
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        envFrom:
        - configMapRef:
            name: app-config-prod  # Для разработки изменить на app-config-dev
```

CLI Examples and Practical Usage

Использование kubectl run

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

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

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

Использование kubectl create

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

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

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

Сложные примеры переменных окружения

Микросервисы с сервис-дискавери

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

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

Многоконтейнерный Pod с общей конфигурацией

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

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

Best Practices

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 подчеркивания 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    # Несогласованное написание
# - name: log-level       # Несогласованный разделитель
```

4. Значения по умолчанию и валидация

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: app
    image: myapp:latest
    env:
    - name: PORT
      value: "8080"          # Предоставляйте разумные значения по умолчанию
    - name: LOG_LEVEL
      value: "info"          # Значения по умолчанию должны быть безопасными
    - name: TIMEOUT_SECONDS
      value: "30"            # Включайте единицы измерения в имена
    - name: MAX_RETRIES
      value: "3"             # Ограничивайте количество повторных попыток
```

Menu

Руководства

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

Extended Resources Quotas

Limit Range

Понимание Limit Range

Создание Limit Range с помощью CLI

Pod Security Admission

Режимы безопасности

Стандарты безопасности

Конфигурация

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

Подготовка перед созданием приложения

Настройка ConfigMap

Понимание ConfigMap

Ограничения ConfigMap

ConfigMap и Secret

Создание ConfigMap через веб-консоль

Создание ConfigMap через CLI

Операции

Просмотр, редактирование и удаление через CLI

Способы использования ConfigMap в поде

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

Создание приложений

Создание приложений из образа

Creating applications from Chart

Создание приложений из YAML

Создание приложений из кода

Создание приложений на основе Operator Backed

Создание приложений с помощью CLI

Предварительные требования

Процедура

Пример

Справка

Конфигурация после создания приложения

Настройка HPA

Понимание Horizontal Pod Autoscalers

Предварительные требования

Создание Horizontal Pod Autoscaler

Правила вычисления

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

Эксплуатация и сопровождение

Описание статуса

Приложения

Запуск и остановка приложений

Запуск приложения

Остановка приложения

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 статуса

Управление версиями приложений

Создание снимка версии

Откат к исторической версии

Удаление приложений

Проверки состояния

Понимание проверок состояния

Пример YAML файла

Параметры настройки проверок состояния через веб-консоль

Устранение неполадок при сбоях проб

Наблюдаемость приложений

Мониторинговые панели

Предварительные требования

Панели мониторинга на уровне Namespace

Мониторинг на уровне рабочих нагрузок

Логи

Процедура

События

Процедура

Интерпретация записей событий

Рабочие нагрузки

Deployments

Understanding Deployments

Creating Deployments

Managing Deployments

Troubleshooting by using 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

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

Pod

Введение

Параметры Pod

Удаление Pods

Сценарии использования

Процедура

Контейнер

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

Extended Resources Quotas

Limit Range

Понимание Limit Range

Создание Limit Range с помощью CLI

Pod Security Admission

Режимы безопасности

Стандарты безопасности

Конфигурация

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 пространств имён

Содержание

Понимание пространств имён

Обратитесь к официальной документации Kubernetes: Namespaces ↗

> В Kubernetes пространства имён обеспечивают 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, основанная на пространстве имён, применима только к объектам с пространством имён (например, Deployments, Services и т.д.), но не к объектам, охватывающим весь кластер (например, StorageClass, Nodes, PersistentVolumes и т.д.).

Создание пространств имён с помощью веб-консоли

> Внутри кластера, связанного с проектом, создайте новое пространство имён, соответствующее доступным квотам ресурсов проекта. Новое пространство имён работает в рамках 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, или при добавлении нового Pod или PVC, это будет расходовать квоту, установленную 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 по какому-либо ресурсу 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 настроить только при наличии GPU-ресурсов в кластере.
     - При использовании vGPU также можно задавать квоты по памяти.

Определения единиц GPU:

- **vGPU Units**: 100 виртуальных GPU (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 хранения	Все	Gi	Общая запрашиваемая ёмкость хранения всех Persistent Volume Claims (PVC) в этом пространстве имён не может превышать это значение.
	Storage Class		Общая запрашиваемая ёмкость хранения всех Persistent Volume Claims (PVC), связанных с

Категория	Тип квоты	Значение и единица	Описание
			выбранным StorageClass в этом пространстве имён, не может превышать это значение. **Примечание**: Пожалуйста, заранее выделите StorageClass проекту, к 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 предотвращения проблем с дублированием ресурсов следующие значения не допускаются в качестве типов квот: • limits.cpu • limits.memory • requests.cpu • requests.memory

Категория	Тип квоты	Значение и единица	Описание
			• 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 с помощью меток или дополнить пространство имён дополнительной информацией через аннотации; оба способа можно использовать для фильтрации и сортировки пространств 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 унаследованные политики уровня проекта (например, Resource Quotas), единый мониторинг и централизованный контроль управления.

Важные замечания:

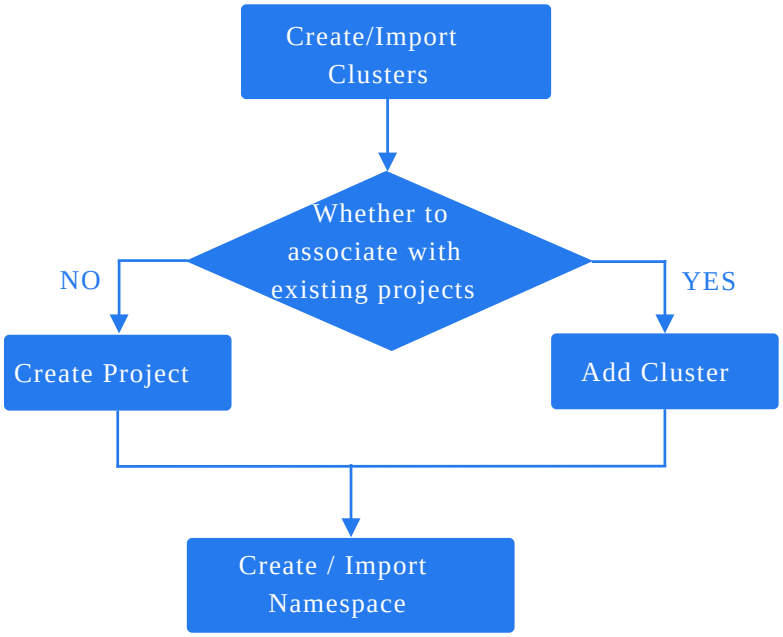
- Namespace может быть связан только с одним проектом в любой момент времени.

- Если Namespace уже связан с проектом, его нельзя импортировать или переназначить в другой проект без предварительного отсоединения от исходного проекта.

Use Cases

Типичные сценарии управления **Namespace** включают:

- При подключении нового **Kubernetes кластера** к платформе можно использовать функцию **Import Namespace** для 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 с помощью скриптов на уровне кластера), должны быть связаны с целевым **проектом** с помощью функции **Import Namespace** для включения управления на уровне платформы, включая видимость и централизованное управление.

Prerequisites

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

Procedure

1. В **Project Management** 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 для конкретного 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 отмечен как `Unlimited`, явная квота не применяется, но использование не может превышать доступную емкость кластера.

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

Поддерживаемые типы квот

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

Примечание:

- Квоты namespace формируются из выделенных проекту ресурсов кластера. Если доступная квота по любому ресурсу равна 0, создание namespace не удастся. 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 через 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 storage class.

Примечание: Убедитесь, что storage class предварительно назначен проекту, содержащему namespace.

Extended Resources Quotas

Расширенные квоты ресурсов определяются через **ConfigMap**. Если ConfigMap отсутствует, категория ресурса не отображается.

Описание полей ConfigMap

Поле	Описание
data.dataType	Тип данных (например, `vGPU`).
data.defaultValue	Значение по умолчанию (пусто = без значения по умолчанию).
data.descriptionEn	Текст подсказки на английском (отображается при наведении на поле).
data.descriptionZh	Текст подсказки на китайском (отображается при наведении на поле).
data.excludeResources	Взаимоисключающие ресурсы (через запятую).
data.group	Группа ресурсов (например, `MPS`).
data.groupI18n	Название группы на английском/китайском для выпадающих списков UI.
data.key	Определяет значение ключа. Словарь конфигурации может описывать только один ключ.
data.labelEn/data.labelZh	Название ресурса на английском/китайском, которое можно просматривать и выбирать в выпадающих списках, соответствующих типам квот. Это поле выполняет ту же функцию, что и data.groupI18n, но применяется только когда у одного ресурса одно значение, обеспечивая совместимость с устаревшей версией словаря конфигурации (ConfigMap).
data.limits	Указывает, нужно ли настраивать лимиты для ресурсов. Допустимые значения: disabled — лимиты

Поле	Описание
	не настраиваются, required — обязательный ввод, optional — необязательный ввод.
data.requests	Указывает, нужно ли настраивать requests для ресурсов. Допустимые значения: disabled — requests не настраиваются, required — обязательный ввод, optional — необязательный ввод, fromLimits — использовать такую же конфигурацию, как для limits.
data.relatedResources	Связанные ресурсы. Поле зарезервировано и в настоящее время не используется.
data.resourceUnit	Единица ресурса (например, `cores`, `GiB`). Ввод на китайском не поддерживается.
data.runtimeClassName	Класс runtime (по умолчанию: `nvidia` для GPU).
metadata.labels	Обязательные метки: • `features.cpaas.io/type: CustomResourceLimitation` • `features.cpaas.io/group: <groupName>` • `features.cpaas.io/enabled`: `true` или `false`, метка обязательна и указывает, включена ли функция, по умолчанию true.
metadata.name	Формат: `cf-crl-<*groupName*>-<*name*>`, где • cf-crl — фиксированное поле, менять нельзя. • groupName — имя соответствующей группы ресурсов, например gpu-manager, galaxy и т.д. • name — имя ресурса: • Имя ресурса может быть стандартным типом ресурса, например cpu, memory, pods и т.д. Стандартные имена должны соответствовать правилам квалифицированных имен Kubernetes

Поле	Описание
	и существовать среди определённых стандартных типов ресурсов Kubernetes. • Имя ресурса также может быть специальным типом ресурса, начинающимся с определённых префиксов, например: `hugepages-` или `requests.hugepages-`.
metadata.namespace	Должен быть `kube-public`

Другие квоты

Формат имен пользовательских квот должен соответствовать следующим требованиям:

- Если имя пользовательской квоты не содержит слэш (/): оно должно начинаться и заканчиваться буквой или цифрой, может содержать буквы, цифры, дефисы (-), подчеркивания (_) или точки (.), формируя квалифицированное имя длиной не более 63 символов.
- Если имя пользовательской квоты содержит слэш (/): 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 только из дефисов, максимальная длина — 63 символа.
  - Поддомен: расширяет правила метки, позволяя соединять несколько меток 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, лимиты и максимальные значения для контейнеров или Pod, созданных после создания или обновления LimitRange, при этом постоянно контролируя использование ресурсов контейнером, чтобы гарантировать, что никакие ресурсы не превышают определённые максимальные значения в пределах namespace.

> 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 - 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 максимумы ресурсов и соотношения, определённые в любом LimitRange, присутствующем в namespace.

Включает следующие настройки

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

Создание Limit Range с помощью CLI

Примеры YAML файлов

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

Создание через YAML файл

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

Создание напрямую через командную строку

```
kubectl create limitrange example-limitrange --namespace=example --
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 соблюдение 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 несоответствующих Pods.	Производственные среды, требующие строгого соблюдения безопасности.
Audit	Позволяет создавать Pods, но регистрирует нарушения в логах.	Мониторинг и анализ инцидентов безопасности без блокировки рабочих нагрузок.

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

Основные замечания:

- Режим **Enforce** действует только на Pods (например, отклоняет Pods, но разрешает ресурсы, не являющиеся Pods, такие как Deployments).
- Режимы **Audit** и **Warn** применяются как к Pods, 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 возможности Linux.

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']
```

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, улучшая 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 контейнеры, исходя из этих запросов.
- Коэффициент Overcommit: Лимиты / Запросы. Эта настройка определяет допустимый диапазон этого коэффициента в namespace, балансируя гарантии ресурсов и предотвращая их чрезмерное потребление.

Основные возможности

- Повышение плотности использования ресурсов и стабильности приложений в namespace за счёт установки подходящего коэффициента overcommit 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 необходимо отключить oversubscription на уровне кластера/namespace.

Порядок действий

1. Перейдите в **Project Management** и откройте **Namespaces** > **Список Namespace**.
2. Нажмите на целевой ***Namespace name***.
3. Нажмите **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, namespace по умолчанию будет 4. • Запросы контейнера = лимит / коэффициент кластера. • Если лимит не задан, используется квота контейнера по умолчанию для namespace.
Custom	• Установить коэффициент, специфичный для namespace (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. Во вкладке **Namespace Members** нажмите **Import Members**.
5. Следуйте инструкциям ниже, чтобы импортировать всех или некоторых пользователей из списка в пространство имён.

   TIP

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

   - 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 диалога и выберите роль для назначения.
     2. Нажмите **Import All**.
   - Импортировать одного или нескольких пользователей из списка в качестве участников пространства имён.
     1. Отметьте чекбокс перед именем пользователя/отображаемым именем для выбора одного или нескольких пользователей.
     2. Нажмите на выпадающий список справа от пункта **Set Role** внизу диалога 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 на платформу, так и не вошедших.

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. Во вкладке **Namespace Members** нажмите **Add Member**.
5. В поле ввода **Username** введите имя пользователя существующего аккаунта сторонней платформы, поддерживаемой платформой.

   Примечание: 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 участник появится в списке участников пространства имён.
   Одновременно в списке пользователей (**Platform Management > User Management**) вы сможете увидеть этого пользователя. До того, как пользователь успешно войдёт или будет синхронизирован с платформой, источник будет `-`, и его можно удалить; после успешного входа или синхронизации платформа зафиксирует источник аккаунта и отобразит его в списке пользователей.

Удаление участников

Удалите указанных участников пространства имён и удалите их связанные роли для отзыва 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. Во вкладке **Namespace Members** нажмите ⋮ справа от записи участника, которого нужно удалить, и выберите **Remove**.

5. Нажмите **Remove**.

Обновление Namespaces

Содержание

Обновление Quotas

> Resource Quota

Обновление Resource Quota через веб-консоль

1. Перейдите в **Project Management**, затем в левой боковой панели выберите **Namespaces** > **Namespace** List.
2. Нажмите на целевое ***имя namespace***.
3. Нажмите **Actions** > **Update Quota**.
4. Отрегулируйте квоты ресурсов (CPU, Memory, Pods и т.д.) и нажмите **Update**.

Обновление Resource Quota через CLI

> Resource Quota YAML file example

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

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

Обновление Container LimitRanges

> Limit Range

Обновление LimitRange через веб-консоль

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

Обновление LimitRange через CLI

> Limit Range YAML file example

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

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

Обновление Pod Security Admission

> Pod Security Admission

Обновление Pod Security Admission через веб-консоль

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

Обновление Pod Security Admission через CLI

> Update Pod Security Admission CLI command

Удаление/Исключение Namespaces

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

Содержание

Удаление Namespaces

Удалить Namespace: Навсегда удаляет namespace и все ресурсы в нем (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

Подготовка перед созданием приложения

Настройка ConfigMap

Понимание ConfigMap

Ограничения ConfigMap

ConfigMap и Secret

Создание ConfigMap через веб-консоль

Создание ConfigMap через CLI

Операции

Просмотр, редактирование и удаление через CLI

Способы использования ConfigMap в поде

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 описывается, что такое ConfigMap, а также как создавать и использовать их.

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

```yaml
# 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 можно будет использовать в подах.
- Контроллеры могут быть написаны с учетом отсутствия данных конфигурации. Рекомендуется рассматривать каждый компонент, использующий ConfigMap, индивидуально.
- Объекты `ConfigMap` находятся в рамках проекта.
- 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` или `--config` узла OpenShift Container Platform, а также через REST API, так как это не стандартные способы создания подов.

ConfigMap и Secret

Особенность	ConfigMap	Secret
Тип данных	Несекретные	Секретные (например, пароли)
Кодировка	Обычный текст	Base64-кодировка
Сценарии использования	Конфигурации, флаги	Пароли, токены

Создание ConfigMap через веб-консоль

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

Параметр	Описание
Entries	Относится к парам `ключ:значение`, поддерживает методы **Добавить** и **Импортировать**. • **Добавить**: можно добавлять элементы конфигурации по одному или вставить одну или несколько строк с парами key=value в поле ввода ключа для массового добавления. • **Импортировать**: импортировать текстовый файл размером не более 1M. Имя файла используется как ключ, содержимое файла — как значение, заполняя элемент конфигурации.
Binary Entries	Относится к бинарным файлам размером не более 1M. Имя файла используется как ключ, содержимое файла — как значение, заполняя элемент конфигурации. **Примечание**: После создания ConfigMap импортированные файлы нельзя изменить.

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 через 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 выбрать **Actions** в правом верхнем углу страницы деталей, 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.

Операция	Описание
Обновить	• После добавления или обновления ConfigMap все рабочие нагрузки, которые ссылались на этот ConfigMap (или его элементы конфигурации) через переменные окружения, должны пересоздать свои поды, чтобы новые настройки вступили 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

Операция	Описание
	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 поде

В виде переменных окружения

```
envFrom:
  - configMapRef:
      name: app-config
```

Каждый ключ становится переменной окружения в контейнере.

В виде файлов в томе

```yaml
volumes:
  - name: config-volume
    configMap:
      name: app-config

volumeMounts:
  - name: config-volume
    mountPath: /etc/config
```

Каждый ключ — это файл в каталоге `/etc/config`, а содержимое файла — значение.

В виде отдельных переменных окружения

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

Настройка Secrets

Содержание

Понимание Secrets

В Kubernetes (k8s) Secret — это базовый объект, предназначенный для хранения и управления конфиденциальной информацией, такой как пароли, OAuth-токены, SSH-ключи, TLS-сертификаты и API-ключи. Его основная задача — предотвратить прямое внедрение чувствительных данных в определения Pod или образы контейнеров, что повышает безопасность и портативность.

Secrets похожи на ConfigMaps, но предназначены специально для конфиденциальных данных. Обычно они хранятся в виде base64-кодированных значений и могут использоваться подами различными способами, включая монтирование в виде томов или предоставление в виде переменных окружения.

Характеристики использования

- **Повышенная безопасность**: По сравнению с конфигурационными картами в открытом виде (Kubernetes ConfigMap), Secrets обеспечивают лучшую защиту, храня чувствительные данные в Base64-кодировке. Этот механизм в сочетании с возможностями Kubernetes по контролю доступа значительно снижает риск утечки данных.
- **Гибкость и управление**: Использование Secrets предоставляет более безопасный и гибкий подход, чем жесткое кодирование конфиденциальной информации непосредственно в файлах определения Pod или образах контейнеров. Такое разделение упрощает управление и изменение чувствительных данных без необходимости менять код приложения или образы контейнеров.

Поддерживаемые типы

Kubernetes поддерживает различные типы Secrets, каждый из которых предназначен для конкретных сценариев использования. Обычно платформа поддерживает следующие типы:

- **Opaque**: Универсальный тип Secret для хранения произвольных пар ключ-значение с конфиденциальными данными, такими как пароли или API-ключи.
- **TLS**: Специально предназначен для хранения сертификатов и приватных ключей TLS (Transport Layer Security), часто используемых для HTTPS-соединений и безопасного ingress.
- **SSH Key**: Используется для хранения приватных SSH-ключей, например, для безопасного доступа к Git-репозиториям или другим сервисам с поддержкой SSH.
- **SSH Authentication (kubernetes.io/ssh-auth)**: Хранит данные аутентификации для передачи 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 как файлы в том пода, позволяя приложениям считывать конфиденциальные данные по заданному пути.

Примечание: Экземпляры Pod в рамках workload могут ссылаться только на Secrets в том же namespace. Для расширенного использования и конфигураций YAML 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 объекта 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, webhook и др.)

```
kubectl create secret tls tls-secret \
--cert=path/to/tls.crt \
--key=path/to/tls.key
```

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: <base64>
  tls.key: <base64>
```

Создание Secret через веб-консоль

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Configuration** > **Secrets**.
3. Нажмите **Create Secret**.
4. Настройте параметры.

   Примечание: В форме ввода конфиденциальные данные, такие как имя пользователя и пароль, автоматически кодируются в формат Base64 перед сохранением в Secret. Преобразованные данные можно просмотреть в YAML-виде.
5. Нажмите **Create**.

Как использовать Secret в Pod

В виде переменных окружения

```yaml
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 workloads для нативных приложений в том же namespace можно 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, workloads, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, необходимо пересобрать Pods, чтобы новая конфигурация вступила в силу.
Удалить	• После удаления Secret workloads, которые ссылались на этот Secret (или его элементы конфигурации) через переменные окружения, могут столкнуться с проблемами из-за отсутствия источника ссылки при пересборке Pods. • Пожалуйста, не удаляйте Secrets, автоматически созданные платформой, так как это может привести к некорректной работе

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

Menu

Создание приложений

Создание приложений из образа

Предварительные требования

Процедура 1 — Workloads

Процедура 2 — Services

Процедура 3 — Ingress

Операции управления приложением

Справочная информация

Creating applications from Chart

Precautions

Prerequisites

Procedure

Status Analysis Reference

Создание приложений из YAML

Меры предосторожности

Предварительные требования

Процедура

Создание приложений из кода

Создание приложений на основе Operator Backed

Создание приложений с помощью 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 подтягивать из него в текущем кластере.

Процедура 1 — Workloads

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

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, можно фильтровать образы по **Already Integrated**. Название интеграционного проекта, например, образы (docker-registry-projectname), включает имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

1. Следуйте приведённым ниже инструкциям для настройки соответствующих параметров.

Workload 1 — Настройка базовой информации

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

Параметры	Описание
Model	Выберите workload по необходимости: • **Deployment**: Подробное описание параметров см. в Creating Deployment. • **DaemonSet**: Подробное описание параметров см. в Creating DaemonSet.

Параметры	Описание
	• **StatefulSet**: Подробное описание параметров см. в Creating StatefulSet.
Replicas	Определяет желаемое количество реплик Pod в Deployment (по умолчанию: `1`). Настраивается в зависимости от требований workload.
More > **Update Strategy**	Настраивает стратегию `rollingUpdate` для обновлений без простоя: **Max surge** (`maxSurge`): • Максимальное число Pod, превышающее желаемое количество реплик во время обновления. • Принимает абсолютные значения (например, `2`) или проценты (например, `20%`). • Расчёт процентов: `ceil(current_replicas × percentage)`. • Пример: 4.1 → `5` при расчёте от 10 реплик. **Max unavailable** (`maxUnavailable`): • Максимальное число Pod, которые могут быть временно недоступны во время обновления. • Процентные значения не могут превышать `100%`. • Расчёт 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 число Pod во время обновления: `10 + 2 = 12`. • `maxUnavailable=3` → Минимальное число доступных Pod: `10 - 3 = 7`. • Это обеспечивает доступность при контролируемом развёртывании.

Workload 2 — Настройка Pod

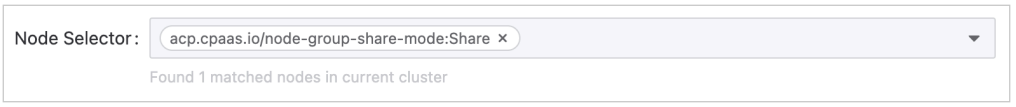
Примечание: В кластерах с разной архитектурой, при развёртывании образов для одной архитектуры, убедитесь в правильной настройке Node Affinity Rules для планирования Pod.

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

Параметры	Описание
Volumes	Монтирование персистентных томов в контейнеры. Поддерживаемые типы томов: `PVC`, `ConfigMap`, `Secret`, `emptyDir`, `hostPath` и др. Для деталей реализации см. Storage Volume Mounting Instructions.
Image Credential	Требуется **только** при подтягивании образов из сторонних регистров (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**: Планирует новые Pod на узлы, где размещены определённые Pod (одинаковый топологический домен). • **Inter-Pod Anti-affinity**: Запрещает совместное размещение новых Pod с определёнными Pod. **Режимы применения**: • **RequiredDuringSchedulingIgnoredDuringExecution**: Pod планируются *только* при выполнении правил. • **PreferredDuringSchedulingIgnoredDuringExecution**: Предпочтение узлам, удовлетворяющим правилам, но допускаются исключения.

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

1. **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	Назначает IP из предопределённого пула подсетей. Если не указано, используется подсеть по умолчанию для namespace.
Static IP Address	Привязывает постоянные IP-адреса 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	Назначает фиксированные 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** настройте соответствующую информацию согласно следующим инструкциям.

Параметры	Описание
Resource Requests & Limits	• **Requests**: Минимальные CPU/память, необходимые для работы контейнера. • **Limits**: Максимальные CPU/память, разрешённые во время выполнения контейнера. Для определения единиц см. Resource Units. **Коэффициент overcommit namespace**: • **Без коэффициента overcommit**: Если существуют квоты ресурсов namespace: Requests/limits контейнера 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. • При отсутствии квот namespace — нет ограничений ресурсов контейнера.
Extended Resources	Настройка расширенных ресурсов, доступных в кластере (например, vGPU, pGPU).
Volume Mount	Конфигурация персистентного хранилища. См. Storage Volume Mounting Instructions. **Операции**: • Для существующих томов pod: нажмите **Add** • Если томов pod нет: нажмите **Add & Mount** **Параметры**: • `mountPath`: Путь в файловой системе контейнера (например, `/data`) • `subPath`: Относительный путь файла/каталога внутри тома. Для `ConfigMap` / `Secret`: выбор конкретного ключа • `readOnly`: Монтировать только для чтения (по умолчанию: чтение-запись) См. Kubernetes Volumes ↗.
Port	Открытие портов контейнера. **Пример**: Открыть TCP порт `6379` с именем `redis`. **Поля**: • `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**: Вывод `$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/конфигурационных файлов.
More > **Referenced ConfigMap**	Внедрение всего ConfigMap/Secret как переменных окружения. Поддерживаемые типы Secret: `Opaque`, `kubernetes.io/basic-auth`.
More > **Health Checks**	• **Liveness Probe**: Проверка здоровья контейнера (перезапуск при сбое) • **Readiness Probe**: Проверка доступности сервиса (удаление из endpoints при сбое) См. Health Check Parameters.
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`: 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 File**	Исключение определённых логов из сбора (например, `/var/log/aaa.log`).
More > **Execute before Stopping**	Выполнение команд перед завершением контейнера. **Пример**: `echo "stop"` **Примечание**: Время выполнения команды должно быть меньше `terminationGracePeriodSeconds` pod.

2. Нажмите **Add Container** (вверху справа) ИЛИ **Add Init Container**.

   См. Init Containers ↗. Init Container:

   1.1. 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, работающему 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, создаваемого под приложением, — это имя вычислительного компонента. Если тип вычислительного компонента (режим развёртывания) — StatefulSet, рекомендуется не менять имя внутреннего маршрута (имя workload), иначе могут возникнуть проблемы с 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 веб-концепции, такие как URI, имена хостов, пути и др. Концепция Ingress позволяет направлять трафик на разные backend в зависимости от правил, определённых через Kubernetes API. Для подробного описания параметров см. Creating Ingresses. **Примечание**: **Service**, используемый при создании **Ingress** под приложением, должен быть ресурсом, созданным в рамках текущего приложения. При этом убедитесь, что **Service** связан с workload под приложением, иначе обнаружение и доступ к workload не будут работать.

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 (в качестве примера показана стратегия Deployment). Сохраняет текущее количество реплик и конфигурацию развёртывания. • **Force Update**: Запускает полное развёртывание приложения с использованием стратегии обновления каждого компонента. 1. **Сценарии использования**: • Пакетные изменения 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 порты по умолчанию. 2) Сохранившиеся правила маршрутизации ссылаются на

Операция	Описание
	компоненты приложения. Полное прекращение внешнего доступа требует удаления Service. • Удаление Service: Необратимая потеря сетевого подключения к компонентам приложения. Связанные правила Ingress становятся неработоспособными, 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. • **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

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

Тип	Назначение
	ассоциированным PV). Непривязанные PVC приведут к ошибкам создания pod.
ConfigMap	Монтирует полные/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/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 особенностями: • Динамическое выделение • Жизненный цикл связан с pod • Поддержка декларативной конфигурации **Сценарий использования**: временное хранение данных. См. Ephemeral Volumes
Empty Directory	Временное хранилище, общее для контейнеров в одном pod: - Создаётся на узле при старте pod - Удаляется при удалении pod **Сценарий использования**: обмен файлами между контейнерами, временное хранение данных. См. EmptyDir

Тип	Назначение
Host Path	Монтирует директорию хост-машины (должна начинаться с `/`, например, `/volumepath`).

Параметры проверки здоровья

Общие параметры

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

Параметр	Применимые протоколы	Описание
Protocol	HTTP/HTTPS	Протокол проверки здоровья
Port	HTTP/HTTPS/TCP	Целевой порт контейнера для проверки.
Path	HTTP/HTTPS	Путь конечной точки (например, `/healthz`).

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.

Creating applications from Chart

Основываясь на Helm Chart, представляет собой нативный шаблон развертывания приложений. Helm Chart — это набор файлов, определяющих ресурсы Kubernetes, предназначенный для упаковки приложений и облегчения их распространения с возможностями контроля версий. Это обеспечивает беспрепятственный переход между средами, например, миграцию из среды разработки в производственную среду.

Содержание

Precautions

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

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

Prerequisites

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.

Procedure

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, выберите Chart и настройте необходимые параметры, такие как `resources.requests`, `resources.limits` и другие параметры, тесно связанные с Chart.
5. Нажмите **Create**.

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

Status Analysis Reference

Нажмите на ***Application Name***, чтобы отобразить подробный анализ статуса Chart в информации о деталях.

Type	Reason
Initialized	Указывает статус загрузки шаблона Chart.

Type	Reason

	• **True**: Шаблон Chart успешно загружен.
	• **False**: Загрузка шаблона Chart не удалась; конкретную причину ошибки можно посмотреть в столбце сообщения.
	  • `ChartLoadFailed` : Загрузка шаблона Chart не удалась.
	  • `InitializeFailed` : Произошло исключение в процессе инициализации до загрузки Chart.

Validated	Указывает статус проверки прав пользователя, зависимостей и других валидаций для шаблона Chart.
	• **True**: Все проверки прошли успешно.
	• **False**: Есть проверки, которые не прошли; конкретную причину ошибки можно посмотреть в столбце сообщения.
	  • `DependenciesCheckFailed` : Проверка зависимостей Chart не удалась.
	  • `PermissionCheckFailed` : У текущего пользователя нет прав на выполнение операций с некоторыми ресурсами.
	  • `ConsistentNamespaceCheckFailed` : При развертывании приложений через шаблоны в нативных приложениях Chart содержит ресурсы, требующие развертывания в разных namespace.

Synced	Указывает статус развертывания шаблона Chart. • **True**: Шаблон Chart успешно развернут. • **False**: Развертывание шаблона Chart не удалось; в столбце причины указано `ChartSyncFailed` , конкретную причину ошибки можно посмотреть в столбце сообщения.

WARNING

- Если шаблон ссылается на ресурсы из разных 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 за помощью 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 одним кликом из YAML. Этот подход позволяет более гибко настраивать базовую информацию и ресурсы для вашего cloud-native приложения.

Содержание

Меры предосторожности

Когда в кластере присутствуют как Linux, так и Windows узлы, чтобы предотвратить планирование приложения на несовместимых узлах, необходимо настроить выбор узла. Например:

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

Предварительные требования

Убедитесь, что образы, определённые в YAML, могут быть загружены в текущем кластере. Это можно проверить 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 построения контейнерных образов непосредственно из исходного кода. Такой подход стандартизирует и автоматизирует процесс сборки приложения, позволяя разработчикам сосредоточиться на разработке исходного кода, не беспокоясь о деталях контейнеризации.

Содержание

Предварительные 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**.

Создание приложений на основе Operator Backed

Приложения на основе Operator backed — это наборы ресурсов, предоставляемые Operator. На основе этих приложений, поддерживаемых Operator, вы можете быстро развернуть компонентное приложение и использовать возможности Operator для автоматизации всего жизненного цикла управления приложением.

Содержание

Процедура

1. В **Container Platform** перейдите в **Applications** > **Applications** в левой боковой панели.
2. Нажмите **Create**.
3. Выберите **Create from Catalog** в качестве способа создания.
4. Выберите экземпляр, поддерживаемый Operator, и настройте **Custom Resource Parameters**. Выберите экземпляр приложения, управляемого Operator, и настройте его спецификации Custom Resource (CR) в манифесте CR, включая:
   - `spec.resources.limits` (ограничения ресурсов на уровне контейнера).
   - `spec.resourceQuota` (политики квот, определённые Operator). Другие параметры, специфичные для 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 (reconciliation). Завершение может занять несколько минут в зависимости от состояния кластера.

Устранение неполадок

Если создание ресурсов 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-ресурсов:

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

3. Повторите создание после проверки готовности CRD/Operator:

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

Создание приложений с помощью CLI

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

CLI-инструмент облегчает развертывание приложений, интеллектуально интерпретируя входные артефакты (образы, Chart и т. д.) и создавая соответствующие объекты Kubernetes API. Создаваемые ресурсы зависят от типа входных данных:

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

Содержание

Предварительные требования

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

Процедура

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

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

4. Просматривайте вывод команд в реальном времени

Пример

YAML

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

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

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

Команды kubectl

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

Справка

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

Menu

Конфигурация после создания приложения

Настройка HPA

Понимание Horizontal Pod Autoscalers

Предварительные требования

Создание Horizontal Pod Autoscaler

Правила вычисления

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

Настройка HPA

HPA (Horizontal Pod Autoscaler) автоматически масштабирует количество подов вверх или вниз на основе заданных политик и метрик, позволяя приложениям справляться с резкими 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 и масштабирование происходят с регулярным интервалом, но может пройти от одной до двух минут, прежде чем метрики станут доступны.

Для replication controllers 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 автоскейлирование во время пиков 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 имеют 0% использования CPU при масштабировании вверх и 100% CPU при масштабировании вниз. Это обеспечивает большую стабильность при принятии решений HPA. Для использования этой функции необходимо настроить readiness checks, чтобы определить, готов ли новый под к использованию.

Как работает HPA?

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

HPA работает как управляющий цикл с периодом синхронизации по умолчанию 15 секунд. В течение этого периода controller manager опрашивает использование CPU, памяти или обоих, согласно конфигурации 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 усредняет использование по всем целевым подам и формирует коэффициент, который используется для масштабирования желаемого количества реплик.

Поддерживаемые метрики

Следующие метрики поддерживаются горизонтальными автоскейлерами подов:

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

> **Важно**: Для автоскейлинга на основе памяти использование памяти должно пропорционально увеличиваться и уменьшаться с количеством реплик. В среднем:
>
> - Увеличение количества реплик должно приводить к общему снижению использования памяти (working set) на под.
> - Уменьшение количества реплик должно приводить к общему увеличению использования памяти на под.
> - Используйте платформу для проверки поведения памяти вашего приложения и убедитесь, что оно соответствует этим требованиям перед использованием автоскейлинга на основе памяти.

Предварительные требования

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

Создание Horizontal Pod Autoscaler

Использование CLI

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

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

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

(1) Используйте API autoscaling/v2.

(2) Имя ресурса HPA.

3. Имя деплоя для масштабирования.
4. Максимальное количество реплик для масштабирования вверх.
5. Минимальное количество реплик для поддержания.
6. Укажите версию API объекта для масштабирования.
7. Укажите тип объекта. Объект должен быть Deployment, ReplicaSet или StatefulSet.
8. Целевой ресурс, к которому применяется HPA.
9. Целевой процент использования CPU, 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 выберите **Workloads** > **Deployments**.
3. Кликните по ***Имя деплоя***.
4. Прокрутите вниз до раздела **Elastic Scaling** и нажмите **Update** справа.
5. Выберите **Horizontal Scaling** и заполните конфигурацию политики.

Параметр	Описание
Pod Count	После успешного создания деплоя необходимо оценить **Минимальное количество подов**, соответствующее известным и регулярным изменениям бизнес-объема, а также **Максимальное количество подов**, которое может поддерживаться квотой namespace при высокой нагрузке. Максимальное и 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-требований.
Trigger Policy	Перечислите **Метрики**, чувствительные к изменениям бизнеса, и их **Целевые пороги**, которые запускают действия 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

Параметр	Описание
	к деплоям в нативных приложениях, и их необходимо предварительно добавить custom metrics .
Scale Up/Down Step (Alpha)	Для бизнесов с особыми требованиями к скорости масштабирования можно постепенно адаптироваться к изменениям объема, задавая **Шаг масштабирования вверх** или **Шаг масштабирования вниз**. Для шага масштабирования вниз можно настроить **Окно стабильности**, по умолчанию 300 секунд, что означает необходимость ожидания 300 секунд перед выполнением действий по уменьшению масштаба.

6. Нажмите **Update**.

Использование пользовательских метрик для HPA

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

Требования

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

Традиционный (Core Metrics) HPA

Традиционный HPA поддерживает метрики использования CPU и памяти для динамического изменения количества подов, 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` задаёт 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 пода), 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 будет запускаться только при отклонении фактического значения за пределы ±10% от целевого порога.
- Масштабирование вниз может повлиять на текущие бизнес-процессы; будьте осторожны.

Правила вычисления

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

- Целевое количество подов для одной политики: ceil[(sum(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 сейчас 3 пода с использованием CPU 80%, 80% и 90%, а установлен порог CPU 60%, то по формуле количество подов будет автоматически скорректировано до: ceil[(80%+80%+90%)/60%] = ceil 4.1 = 5 подов.

  Примечание:

  - Если рассчитанное целевое количество подов превышает установленное **Максимальное количество подов** (например, *4*), платформа масштабирует только до 4 подов. Если после изменения максимума метрики остаются высокими, возможно, потребуется использовать альтернативные методы масштабирования, например, увеличить квоту подов в namespace или добавить аппаратные ресурсы.
  - Если рассчитанное целевое количество подов (в предыдущем примере *5*) меньше количества подов, рассчитанного с учётом **Шага масштабирования вверх**

(например, *10*), платформа масштабирует только до 5 подов.

- Целевое количество подов для нескольких политик: выбирается максимальное значение из результатов расчёта каждой политики.

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 в соответствии с вашими бизнес-требованиями, обеспечивая достаточность ресурсов для подов и повышая эффективность использования ресурсов кластера.

Содержание

Понимание VerticalPodAutoscalers

Вы можете создать VerticalPodAutoscaler, который будет рекомендовать или автоматически обновлять запросы и лимиты ресурсов CPU и памяти для ваших подов на основе их исторических шаблонов использования.

После создания VerticalPodAutoscaler платформа начинает отслеживать 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 приводит к более эффективному использованию ресурсов в кластере.

Как работает VPA?

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

VPA работает, постоянно отслеживая использование ресурсов 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 дает хорошие результаты, но при недостатке ресурсов в кластере 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 возможности

VerticalPodAutoscaler предоставляет рекомендации по ресурсам на основе исторических шаблонов использования, что позволяет оптимизировать конфигурации CPU и памяти ваших подов.

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

Предварительные требования

- 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 страницу списка **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 интерфейс командной строки, определив YAML-файл и используя команду `kubectl create`. В следующем примере показано вертикальное автоскейлирование подов для объекта Deployment:

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

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

(1) Используйте API autoscaling.k8s.io/v1.

(2) Имя VPA.

(3) Укажите целевой объект рабочей нагрузки. VPA использует селектор рабочей нагрузки для поиска подов, которым требуется корректировка ресурсов. Поддерживаемые типы рабочих нагрузок включают DaemonSet, Deployment, ReplicaSet, StatefulSet, ReplicationController, Job и CronJob.

(4) Укажите версию API объекта для масштабирования.

(5) Укажите тип объекта.

(6) Целевой ресурс, к которому применяется VPA.

(7) Политика обновления, определяющая, как VPA применяет рекомендации. Параметр updateMode может принимать значения:

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

- Recreate: Автоматически устанавливает запросы ресурсов при создании подов и эвакуирует текущие поды для обновления до рекомендуемых значений. 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, которая может 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 (частичный):

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

Использование веб-консоли

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

Параметр	Описание
Режим масштабирования	В настоящее время поддерживается режим **Manual Scaling**, который предоставляет рекомендуемые конфигурации ресурсов на основе анализа прошлых данных об использовании ресурсов. Вы можете вручную корректировать значения согласно рекомендациям. Корректировки приведут к пересозданию и перезапуску подов, поэтому выбирайте подходящее время, чтобы не повлиять на работающие приложения. Обычно после того, как поды работают более 8 дней, рекомендуемые значения становятся точными. Обратите внимание, что при недостатке ресурсов в кластере масштабирование может привести к состоянию

Параметр	Описание
	Pending у подов. Убедитесь, что в кластере достаточно ресурсов или установлены разумные квоты, либо настройте оповещения для мониторинга условий масштабирования.
Целевой контейнер	По умолчанию выбирается 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 сохранение минимального количества доступных 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 об использовании, поэтому может потребоваться несколько дней работы подов, прежде чем рекомендации станут точными.
- При применении рекомендаций VPA в режиме Auto происходит пересоздание подов, что может вызвать временные перебои в работе приложения.

Последующие действия

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

Настройка CronHPA

Для бессостоя́тельных приложений с периодическими колебаниями бизнес-активности CronHPA (Cron Horizontal Pod Autoscaler) поддерживает регулирование количества подов на основе заданных 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, которое должно работать в определённое время согласно расписанию. Это позволяет подготовиться к предсказуемым пиковым нагрузкам или снизить использование ресурсов в непиковые часы.

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, определяя одно или несколько правил расписания, каждое из которых задаёт время (в формате crontab) и целевое количество реплик. Когда наступает запланированное время, CronHPA изменяет количество подов в соответствии с указанной целью, независимо от текущей загрузки 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 непиковые часы.

CronHPA постоянно сравнивает текущее время с заданными расписаниями. Когда наступает запланированное время, контроллер изменяет количество подов, чтобы соответствовать 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 через интерфейс командной строки, определив YAML-файл и используя команду `kubectl create`. В следующем примере показано плановое масштабирование для объекта Deployment:

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

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

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

(2) Имя ресурса CronHPA.

(3) Имя деплоймента для масштабирования.

(4) Укажите API версию объекта для масштабирования.

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

(6) Целевой ресурс, к которому применяется CronHPA.

(7) Расписание в стандартном формате crontab (минуты часы день месяц день_недели).

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

Этот пример настраивает деплоймент следующим образом:

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

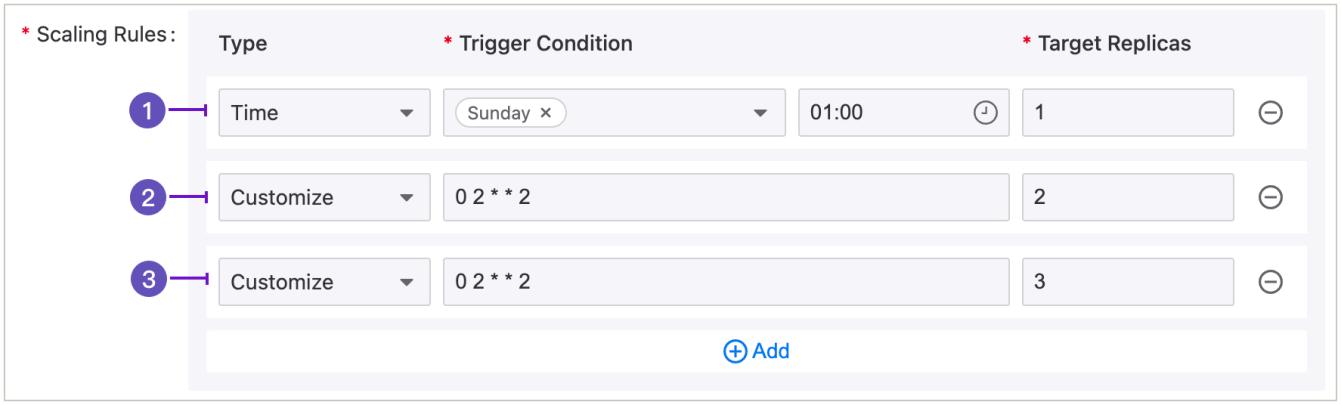
2. Примените YAML-файл для создания CronHPA:

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

Использование веб-консоли

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

Объяснение правил расписания



1. Указывает, что начиная с 01:00 каждого понедельника будет поддерживаться только 1 под.

2. Указывает, что начиная с 02:00 каждого вторника будет поддерживаться только 2 пода.

3. Указывает, что начиная с 02:00 каждого вторника будет поддерживаться 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 (`минуты часы 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.

Menu

Эксплуатация и сопровождение

Описание статуса

Приложения

Запуск и остановка приложений

Запуск приложения

Остановка приложения

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 статуса

Управление версиями приложений

Создание снимка версии

Откат к исторической версии

Удаление приложений

Проверки состояния

Понимание проверок состояния

Пример YAML файла

Параметры настройки проверок состояния через веб-консоль

Устранение неполадок при сбоях проб

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, 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 вычислительных компонентов 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.

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**.

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 неймспейсами с помощью `kubectl apply -f`

Импортированные в приложение ресурсы получают следующие преимущества:

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

Содержание

Импорт ресурсов

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

1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Application Management** > **Native Applications**.
3. Кликните 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, можно быстро найти нужный тип по ключевым словам.
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 или пакетное удаление ресурсов из приложения лишь разрывает связь ресурсов с приложением 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 сможет получить доступ к *Deployment D* через *Service S*.

1. Перейдите в **Container Platform**.

2. В левой навигационной панели выберите **Application Management** > **Native Applications**.

3. Кликните на ***Название приложения***.

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

5. Справа от ресурса нажмите **Remove** для удаления; либо выберите несколько ресурсов 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 нажмите **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 способ экспорта и настройте соответствующую информацию согласно следующим инструкциям.

   - Экспорт Helm Charts в репозиторий шаблонов с правами управления

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

Параметр	Описание
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 её 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, 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 способа экспорта, вы можете ознакомиться с разделом Создание 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 нажмите **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**, после чего можно экспортировать упрощённый YAML-файл, который можно напрямую развернуть в других средах.
6. Нажмите **Export**.

Метод 2

1. Зайдите в **Container Platform**.
2. В левой навигационной панели нажмите **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 **Preview Simplified YAML** не выбран, 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	Выберите **Preview Simplified YAML**, после чего можно просмотреть и экспортировать упрощённый YAML-файл, который можно напрямую развернуть в других средах.

5. Нажмите **Export**.

Последующие действия

После экспорта упрощённого YAML вы можете ознакомиться с разделом Создание 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 нажмите **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 способ экспорта и настройте соответствующую информацию согласно следующим инструкциям.

   - Экспорт YAML в репозиторий кода:

Параметр	Описание
Target Location	Выберите **Code Repository** для прямой синхронизации YAML-файла в указанный Git-репозиторий кода. 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 отмечен **Submit PR (Pull Request)**, платформа создаст новую ветку и отправит Pull Request.
	  • Если отмечен **Automatically delete source branch after merging PR**, исходная ветка будет автоматически удалена после слияния PR в Git-репозитории.

Параметр	Описание
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.
Commit Message	Заполните информацию коммита для идентификации содержимого этого коммита.
Preview	Просмотр YAML-файла для отправки и сравнение отличий с существующим YAML в репозитории кода, отображаемое 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 нескольких кластерах.

6. Нажмите **Export**.

Последующие действия

После экспорта YAML в Git-репозиторий кода вы можете ознакомиться с разделом Создание GitOps приложений для создания GitOps-приложения типа `Custom Application` в нескольких кластерах.

Обновление и удаление Chart-приложений

В связи с пересечением функциональности 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 ваши успешно развернутые шаблонные приложения до нативных приложений.

Содержание

Важные замечания

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

Предварительные требования

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

Описание анализа статуса

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

Тип	Причина

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

Validated	Указывает состояние проверки прав пользователя и зависимостей для шаблона Chart.
	• Если статус True, это означает, что все проверки прошли успешно.
	• Если статус False, это означает, что некоторые проверки не прошли, а причину неудачи можно посмотреть в столбце сообщения.
	  • DependenciesCheckFailed: проверка зависимостей Chart не удалась.
	  • PermissionCheckFailed: у текущего пользователя 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, при обновлении приложения через API-вызовы, вы можете вручную создать снимок версии для фиксации изменений.

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

Содержание

Создание снимка версии

Порядок действий

1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **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 распространять приложение, что позволяет распространять снимок версии приложения в виде Chart, облегчая быструю развертку одного и того же приложения на нескольких кластерах и пространствах имён платформы.

Откат к исторической версии

Выполните откат конфигурации текущего приложения к исторической версии.

Порядок действий

1. Перейдите в **Container Platform**.

2. В левой навигационной панели нажмите **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 критически важным механизмом для обеспечения высокой доступности и устойчивости ваших приложений. Kubernetes использует эти пробы для определения состояния здоровья и готовности ваших Pod-ов, позволяя системе предпринимать соответствующие действия, такие как перезапуск контейнеров или маршрутизация трафика. Без правильной настройки проверок состояния Kubernetes не сможет надежно управлять жизненным циклом вашего приложения, что может привести к ухудшению качества сервиса или сбоям.

Kubernetes предлагает три типа проб:

- `livenessProbe`: Определяет, запущен ли контейнер. Если liveness probe не проходит, Kubernetes завершит Pod и перезапустит его согласно политике перезапуска.
- `readinessProbe`: Определяет, готов ли контейнер обслуживать трафик. Если readiness probe не проходит, Endpoint Controller удалит Pod из списка Endpoint-ов Service до тех пор, пока проба не пройдет успешно.
- `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.

- **Сценарии 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.

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

- **Сценарии 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 его перезапустить. При сбое Kubernetes перезапустит контейнер.
  - **Readiness**: Если приложение временно не может обслуживать трафик (например, устанавливает соединение с базой данных), но может восстановиться без перезапуска, используйте Readiness Probe. Это предотвращает 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 эндпоинты проб легковесны и выполняются быстро. Они не должны включать тяжелые вычисления или внешние зависимости (например, вызовы к базе данных), которые могут сделать пробу ненадежной.
- **Содержательные проверки**: Проверки должны реально отражать состояние здоровья и готовности приложения, а не просто факт работы процесса. Например, для веб-сервера проверяйте возможность отдачи базовой страницы, а не только открытость порта.
- **Настройка initialDelaySeconds**: Устанавливайте initialDelaySeconds так, чтобы дать приложению достаточно времени на запуск перед первой проверкой.
- **Настройка periodSeconds и failureThreshold**: Балансируйте между быстрым обнаружением сбоев и предотвращением ложных срабатываний. Слишком частые пробы или слишком низкий failureThreshold могут привести к ненужным перезапускам или состоянию "не готов".

- **Логи для отладки**: Убедитесь, что ваше приложение логирует понятные сообщения, связанные с вызовами эндпоинтов проверок и внутренним состоянием, чтобы облегчить отладку сбоев проб.
- **Комбинирование проб**: Часто все три пробы (Liveness, Readiness, Startup) используются вместе для эффективного управления жизненным циклом приложения.

Пример YAML файла

```yaml
spec:
  template:
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2 # Container image
          ports:
            - containerPort: 80 # Container exposed port
          startupProbe:
            httpGet:
              path: /startup-check
              port: 8080
            initialDelaySeconds: 0 # Обычно 0 для startup проб 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-а перед проверкой
            periodSeconds: 10 # Проверка каждые 10 секунд
            timeoutSeconds: 5 # Таймаут через 5 секунд
            failureThreshold: 3 # Считается нездоровым после 3 последовательных сбоев
          readinessProbe:
            httpGet:
              path: /ready
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 10
            timeoutSeconds: 5
            failureThreshold: 3
```

Параметры настройки проверок состояния через веб-консоль

Общие параметры

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

Если статус Pod-а указывает на проблемы, связанные с пробами, вот как их диагностировать:

Проверьте события Pod-а

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

Ищите события, связанные с LivenessProbe failed, ReadinessProbe failed или StartupProbe failed. Эти события часто содержат конкретные сообщения об ошибках (например, отказ соединения, HTTP 500 ошибка, 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-а 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 YAML Deployment/Pod. Частая ошибка — неправильный 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 неотзывчивости приложения и сбоям проб. Проверьте использование ресурсов Pod-а (`kubectl top pod <pod-name>`) и рассмотрите возможность настройки лимитов/запросов ресурсов.

Проблемы с сетью

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

Menu

Наблюдаемость приложений

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 тренды/рейтинги 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. В **Platform Management** создайте выделенную панель мониторинга, следуя инструкции в разделе Creating Monitoring Dashboard.

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 вызвать нестабильность компонента.
Network Traffic	**Inflow Rate/Outflow Rate**	Сетевой трафик (байт/сек) входящий/исходящий из подов.
Network Packet	**Receiving Rate/Transmit Rate**	Сетевые пакеты (кол-во/сек) принимаемые/отправляемые подами.
Disk Rate	**Read/Write**	Скорость чтения/записи (байт/сек) смонтированных томов на 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. Для создания выделенной панели мониторинга на уровне рабочих нагрузок обратитесь к разделу Add Panel in Custom Dashboard.

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 коренные причины.

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 списка для просмотра соответствующих логов.
Previous Logs	Просмотр логов завершённых контейнеров (доступно при restartCount контейнера > 0).

Операция	Описание
Lines	Настройка размера буфера отображаемых логов: 1k/10k/100k строк.
Wrap Line	Переключение переноса строк для длинных записей лога (включено по умолчанию).
Find	Полнотекстовый поиск с подсветкой совпадений и переходом по нажатию Enter.
Raw	Необработанные потоки логов, напрямую захваченные с интерфейсов контейнерного рантайма (CRI) без форматирования, фильтрации или усечения.
Export	Скачивание необработанных логов.
Full Screen	Клик по усечённой строке для просмотра полного содержимого в модальном окне.

WARNING

- **Обработка усечения**: Записи лога длиной более 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. Всегда используйте функции **Export** или **Raw** для получения полных логов.
- **Политика хранения**: Живые логи подчиняются 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**, чтобы просмотреть подробные записи.

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 повторений.

> **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

Рабочие нагрузки

Deployments

Understanding Deployments

Creating Deployments

Managing Deployments

Troubleshooting by using 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

Deployments

Содержание

Understanding Deployments

Обратитесь к официальной документации Kubernetes: Deployments ↗

> **Deployment** — это ресурс высокого уровня в Kubernetes для управления и обновления реплик Pod в декларативном стиле. Он предоставляет надежный и гибкий способ определить, как должно работать ваше приложение, включая количество поддерживаемых реплик и безопасное выполнение поэтапных обновлений.

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.

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.

Как это работает:

1. Вы определяете желаемое состояние приложения через Deployment (например, какой образ использовать, сколько реплик запускать).

2. Deployment создает ReplicaSet, который следит за тем, чтобы заданное количество Pod было запущено.

3. ReplicaSet создает и управляет фактическими экземплярами Pod.

4. При обновлении Deployment (например, смене версии образа) создается новый ReplicaSet, который постепенно заменяет старые Pod новыми согласно стратегии rolling update, пока все новые Pod не будут запущены, после чего старый ReplicaSet удаляется.

Creating Deployments

Creating a Deployment by using CLI

Prerequisites

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

YAML file example

```yaml
# example-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment # Имя Deployment
  labels:
    app: nginx # Метки для идентификации и выбора
spec:
  replicas: 3 # Желаемое количество реплик Pod
  selector:
    matchLabels:
      app: nginx # Селектор для выбора Pod, 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: # Ограничения и запросы ресурсов
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 200m
              memory: 256Mi
```

Creating a Deployment via YAML

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

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

Creating a Deployment by using web console

Prerequisites

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

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

Procedure - Configure Basic Info

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

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в web console, можно фильтровать образы по **Already Integrated**. **Integration Project Name** — например, образы (docker-registry-projectname), где projectname — имя проекта в web console 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 нагрузки.
More > **Update Strategy**	Настройка стратегии `rollingUpdate` для обновлений без простоя:

Параметры	Описание
	Max surge (`maxSurge`): • Максимальное количество Pod, которое может превышать желаемое число реплик во время обновления. • Принимает абсолютные значения (например, `2`) или проценты (например, `20%`). • Расчет процентов: `ceil(текущее_число_реплик × процент)` . • 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%` . • Расчет процентов: `floor(текущее_число_реплик × процент)` . • 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 одновременно 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: • `maxSurge=2` → Общее количество Pod во время обновления: `10 + 2 = 12` .

Параметры	Описание
	• `maxUnavailable=3` → Минимальное количество доступных Pod: `10 - 3 = 7`. • Это обеспечивает доступность при контролируемом развертывании.

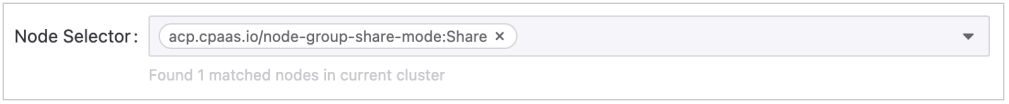
Procedure - Configure Pod

Примечание: В кластерах с разной архитектурой, при развертывании образов для одной архитектуры, убедитесь, 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. Поддерживаются 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 работы 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**	Ограничение Pod узлами с определёнными метками (например, `kubernetes.io/os: linux`). 
More > **Affinity**	Определение тонких правил планирования на основе существующих. **Типы Affinity**: • **Pod Affinity**: Планирование новых Pod на узлы, где уже размещены определённые Pod (одинаковый топологический домен). • **Pod Anti-affinity**: Запрет совместного размещения новых Pod с определёнными Pod. **Режимы применения**: • `requiredDuringSchedulingIgnoredDuringExecution`: Pod планируются *только* если правила удовлетворены. • `preferredDuringSchedulingIgnoredDuringExecution`: Приоритет узлам, удовлетворяющим правилам, но допускаются исключения. **Поля конфигурации**: • `topologyKey`: Метка узла, определяющая топологический домен (по умолчанию: `kubernetes.io/hostname`). • `labelSelector`: Фильтр целевых Pod с помощью запросов по меткам.

3. **Network Configuration**

- Kube-OVN

Параметры	Описание
Bandwidth Limits	Ограничение QoS для сетевого трафика Pod: • **Ограничение скорости исходящего трафика**: Максимальная скорость исходящего трафика (например, `10Mbps`). • **Ограничение скорости входящего трафика**: Максимальная скорость входящего трафика.
Subnet	Назначение IP из предопределенного пула подсетей. 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 ≥ числу реплик Pod.

- Calico

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

Procedure - Configure Containers

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

Параметры	Описание
Resource Requests & Limits	• **Requests**: Минимальные CPU/память, необходимые для работы контейнера. • **Limits**: Максимальные CPU/память, разрешённые во время выполнения контейнера. Для определения единиц смотрите Resource Units. **Коэффициент overcommit namespace**: • **Без коэффициента overcommit**: Если существуют квоты ресурсов namespace: запросы/лимиты контейнера наследуют значения по умолчанию namespace (можно изменить). Без квот namespace: нет значений по умолчанию; настраивается вручную. • **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 запросов. • Отсутствие квот namespace → отсутствие ограничений ресурсов контейнера.
Extended Resources	Настройка расширенных ресурсов, доступных в кластере (например, vGPU, pGPU).
Volume Mounts	Конфигурация постоянного хранилища. Смотрите Storage Volume Mounting Instructions. **Операции**:

Параметры	Описание
	• Для существующих томов pod: нажмите **Add** • Если томов pod нет: нажмите **Add & Mount** **Параметры**: • `mountPath`: путь в файловой системе контейнера (например, `/data`) • `subPath`: относительный путь к файлу/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 Defining Commands↗.
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 полного ConfigMap/Secret как переменных окружения. Поддерживаемые типы Secret: `Opaque` , `kubernetes.io/basic-auth` .
More > **Health Checks**	• **Liveness Probe**: проверка здоровья контейнера (перезапуск при сбое) • **Readiness Probe**: проверка доступности сервиса (удаление из endpoints при сбое) Смотрите Health Check Parameters.
More > **Log Files**	Настройка путей логов: - По умолчанию: сбор `stdout` - Шаблоны файлов: например, `/var/log/*.log` **Требования**: • Драйвер хранения `overlay2` : поддерживается по умолчанию • `devicemapper` : необходимо вручную монтировать EmptyDir в директорию логов • Узлы Windows: обеспечить монтирование родительской директории (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**.

Reference Information

Storage Volume Mounting instructions

Тип	Назначение
Persistent Volume Claim	Привязка существующего PVC для запроса постоянного хранилища. **Примечание**: доступны только привязанные PVC (с ассоциированным PV). Непривязанные PVC приведут к ошибкам создания pod.
ConfigMap	Монтирование полного/частичного ConfigMap как файлов: • Полный ConfigMap: создаёт файлы с именами ключей в каталоге монтирования • Выбор подпути: монтирование конкретного ключа (например, `my.cnf`)

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

Heath Checks

- Пример YAML файла для health checks
- Параметры настройки health checks в web console

Managing Deployments

Managing a Deployment by using CLI

Viewing a Deployment

- Проверьте, что Deployment создан.

```
kubectl get deployments
```

- Получите подробную информацию о вашем Deployment.

```
kubectl describe deployments
```

Updating a Deployment

Выполните следующие шаги для обновления Deployment:

1. Обновим Pods nginx для использования образа nginx:1 .16.1.

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

или используйте команду:

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

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

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

2. Чтобы увидеть статус развертывания, выполните:

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

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

```
kubectl get rs
```

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

```
kubectl get pods
```

Scaling a Deployment

Вы можете масштабировать Deployment с помощью следующей команды:

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

Rolling Back a Deployment

- Предположим, вы допустили опечатку 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 зависло. Вы можете проверить это, посмотрев статус развертывания:

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

Deleting a Deployment

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

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

Managing a Deployment by using web console

Viewing a Deployment

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

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. Нажмите на имя Deployment, чтобы увидеть **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Updating a Deployment

1. В **Container Platform** перейдите в **Workloads** > **Deployments**.
2. Найдите нужный Deployment.
3. В выпадающем меню **Actions** выберите **Update**, чтобы открыть страницу редактирования Deployment.

Deleting a Deployment

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

Troubleshooting by using CLI

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

Check Deployment status

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

Check ReplicaSet status

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

Check Pod status

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

View Logs

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

Enter Pod for debugging

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

Check Health configuration

Убедитесь, что livenessProbe и readinessProbe настроены корректно, а эндпоинты проверки здоровья вашего приложения отвечают правильно. Troubleshooting probe failures

Check Resource Limits

Убедитесь, что запросы и лимиты ресурсов контейнеров разумны и контейнеры не завершаются из-за нехватки ресурсов.

DaemonSets

Содержание

Понимание DaemonSets

Обратитесь к официальной документации Kubernetes: DaemonSets ↗

DaemonSet — это контроллер Kubernetes, который гарантирует, что на всех (или на подмножестве) узлах 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 реплику **Pod** на каждый планируемый **Node**, соответствующий его критериям:
     - Разворачивает ровно **одну реплику 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, с помощью `kubectl cordon`).
   - Узлы со статусом `NotReady`.

- Узлы с несовместимыми **Taints**, для которых в **Pod Template** DaemonSet не настроены соответствующие Tolerations.

Создание DaemonSets

Создание DaemonSet с помощью CLI

Предварительные требования

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

Пример YAML файла

```yaml
# example-daemonSet.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  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 позволяют запускать daemonset 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 с ним 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 с помощью веб-консоли

Предварительные требования

Получите адрес образа. Источником образов может быть репозиторий образов, интегрированный 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 раздел **Workloads** > **DaemonSets** в левой боковой панели.
2. Нажмите **Create DaemonSet**.

3. **Выберите** или **введите** образ и нажмите **Confirm**.

> **INFO**
>
> **Примечание**: При использовании образов из репозитория, интегрированного в веб-консоль, вы можете фильтровать образы по категории **Already Integrated**. **Integration Project Name**, например, images (docker-registry-projectname), включает имя проекта projectname в этой веб-консоли и имя проекта containers в репозитории образов.

В разделе **Basic Info** настройте декларативные параметры для DaemonSet:

Параметры	Описание
More > **Update Strategy**	Настраивает стратегию `rollingUpdate` для обновления Pod DaemonSet без простоя. **Max unavailable** (`maxUnavailable`): Максимальное количество Pod, которые могут быть временно недоступны во время обновления. Принимает абсолютные значения (например, 1) или проценты (например, 10%). **Пример**: Если узлов 10, а maxUnavailable равен 10%, то floor(10 * 0.1) = 1 Pod может быть недоступен. **Примечания:** • **Значения по умолчанию**: Если явно не указано, `maxSurge` по умолчанию 0, а `maxUnavailable` — 1 (или 10%, если указано в процентах). • **Незапущенные Pod**: Pod в состояниях `Pending` или `CrashLoopBackOff` считаются недоступными. • **Одновременные ограничения**: `maxSurge` и `maxUnavailable` не могут оба 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 реплики Pod на всех подходящих узлах, которые соответствуют:
  - Критериям `nodeSelector` (если определены).
  - Конфигурации `tolerations` (позволяющей планировать на узлах с taints).
  - Узлу в состоянии `Ready` и с `Schedulable: true`.
- ❌ Исключённые узлы:
  - Узлы с taint `NoSchedule` (если явно не допускается).
  - Узлы, вручную заблокированные (`kubectl cordon`).
  - Узлы в состояниях `NotReady` или `Unschedulable`.

Управление DaemonSets

Управление DaemonSet с помощью CLI

Просмотр DaemonSet

- Получить сводку всех DaemonSet в namespace:

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

- Получить подробную информацию о конкретном DaemonSet, включая события и статус Pod:

```
kubectl describe daemonset <daemonset-name>
```

Обновление DaemonSet

При изменении **Pod Template** DaemonSet (например, смена образа контейнера или добавление volumeMount), 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 с помощью веб-консоли

Просмотр DaemonSet

1. В **Container Platform** перейдите в **Workloads** > **DaemonSets**.
2. Найдите нужный DaemonSet.
3. Нажмите на имя DaemonSet, чтобы увидеть **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Обновление DaemonSet

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

2. Найдите DaemonSet для обновления.

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

Удаление DaemonSet

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

2. Найдите DaemonSet для удаления.

3. В выпадающем меню **Actions** нажмите кнопку **Delete** в колонке операций и подтвердите.

StatefulSets

Содержание

Понимание StatefulSets

Обратитесь к официальной документации Kubernetes: StatefulSets ↗

StatefulSet — это объект API рабочих нагрузок Kubernetes, предназначенный для управления stateful-приложениями, предоставляя:

- **Стабильную сетевую идентичность**: DNS-имя хоста `<statefulset-name>-<ordinal>.<service-name>.ns.svc.cluster.local`.
- **Стабильное постоянное хранилище**: через `volumeClaimTemplates`.
- **Упорядоченное развертывание/масштабирование**: последовательное создание/удаление Pod-ов: Pod-0 → Pod-1 → Pod-N.
- **Упорядоченные 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: # шаблон Pod-а для StatefulSet
    metadata:
      labels:
        app: nginx # должен совпадать с .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: nginx
          image: registry.k8s.io/nginx-slim:0.24
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
  volumeClaimTemplates: # шаблоны PersistentVolumeClaim (PVC). Каждый Pod получает уникальный PersistentVolume (PV), динамически выделяемый 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 StatefulSet, а также связанных Pod-ов и PVC:
kubectl get statefulset web # Просмотр StatefulSet
kubectl get pods -l app=nginx # Проверка Pod-ов, управляемых этим StatefulSet
kubectl get pvc -l app=nginx # Проверка PVC, созданных volumeClaimTemplates
```

Создание StatefulSet через веб-консоль

Предварительные требования

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

- В первом случае администратор обычно назначает репозиторий образов вашему проекту, и вы можете использовать образы из него. Если нужный репозиторий образов не найден, обратитесь к администратору для выделения.
- Если это репозиторий образов сторонней платформы, убедитесь, что образы можно напрямую загружать 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 индексу 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 выделять постоянное хранилище для каждого Pod-а. Каждая реплика Pod-а в StatefulSet автоматически

Параметры	Описание

	получает собственный уникальный PersistentVolumeClaim (PVC) на основе предопределённых шаблонов.
	• 1. **Динамическое создание PVC**: автоматически создаёт уникальные PVC для каждого Pod-а с шаблоном имени: `<statefulset-name>-<claim-template-name>-<pod-ordinal>`. **Пример**: `web-www-web-0`, `web-www-web-1`.
	• 2. **Режимы доступа**: 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 кластера. 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-а

Раздел **Pod**, пожалуйста, смотрите в Deployment - Configure Pod

Процедура — Настройка контейнеров

Раздел **Containers**, пожалуйста, смотрите в Deployment - Configure Containers

Процедура — Создание

Нажмите **Create**.

Проверка состояния (Health Checks)

- Пример YAML файла для проверки состояния
- Параметры конфигурации проверки состояния в веб-консоли

Управление StatefulSets

Управление StatefulSet с помощью CLI

Просмотр StatefulSet

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

- Проверить, что StatefulSet создан.

```
kubectl get statefulsets
```

- Получить подробную информацию о StatefulSet.

```
kubectl describe statefulsets
```

Масштабирование StatefulSet

- Чтобы изменить количество реплик в существующем 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 можно 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), чтобы предотвратить потерю данных. Чтобы также удалить PVC, сделайте это явно:

```
kubectl delete pvc -l app=<label-selector-for-your-statefulset> # Пример: kubectl delete pvc -l app=nginx
```

Альтернативно, если ваши `volumeClaimTemplates` используют `StorageClass` с политикой `reclaimPolicy` равной `Delete`, PV и базовое хранилище будут удалены автоматически при удалении PVC.

Управление StatefulSet через веб-консоль

Просмотр StatefulSet

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

2. Найдите StatefulSet, который хотите просмотреть.

3. Нажмите на имя StatefulSet, чтобы увидеть **Details**, **Topology**, **Logs**, **Events**, **Monitoring** и др.

Обновление StatefulSet

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

2. Найдите StatefulSet, который хотите обновить.

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

Удаление StatefulSet

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

2. Найдите StatefulSet, который хотите удалить.

3. В выпадающем меню **Actions** нажмите кнопку **Delete** в колонке операций и подтвердите.

CronJobs

Содержание

Понимание CronJobs

Обратитесь к официальной документации Kubernetes:

- CronJobs ↗
- Running Automated Tasks with a CronJob ↗

CronJob определяет задачи, которые выполняются до завершения и затем останавливаются. Они позволяют запускать один и тот же Job несколько раз в соответствии с расписанием.

CronJob — это тип контроллера рабочих нагрузок в Kubernetes. Вы можете создать CronJob через веб-консоль или CLI для периодического или повторяющегося запуска непостоянной программы, такой как запланированные резервные копии, запланированная очистка или запланированная отправка электронной почты.

Создание CronJobs

Создание CronJob с помощью CLI

Предварительные требования

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

Пример YAML файла

```yaml
# example-cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Создание 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, или из сторонних реестров образов.

- Для образов из интегрированного реестра администратор обычно назначает реестр образов вашему проекту, что позволяет использовать образы внутри него. Если нужный реестр образов не найден, обратитесь к администратору для выделения.
- Если используется сторонний реестр образов, убедитесь, что образы можно напрямую загрузить из него в текущем кластере.

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) указывает на имя проекта платформы projectname и имя проекта реестра образов containers.
4. В разделе **Cron Configuration** настройте способ выполнения задачи и связанные параметры.

   Execute Type:

   - **Manual**: Ручное выполнение требует явного ручного запуска для каждого запуска задачи.
   - **Scheduled**: Запланированное выполнение требует настройки следующих параметров расписания:

Параметр	Описание
Schedule	Определяет расписание cron с использованием синтаксиса Crontab ↗. Контроллер CronJob рассчитывает следующее время выполнения на основе выбранного часового пояса. **Примечания**: • Для Kubernetes кластера < v1.25: выбор часового пояса не поддерживается; расписания ДОЛЖНЫ использовать UTC. • Для Kubernetes кластера ≥ v1.25: поддерживается расписание с учётом часового пояса (по умолчанию: локальный часовой пояс пользователя).

Параметр	Описание
Concurrency Policy	Определяет, как обрабатываются параллельные запуски 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 Pod'ов. 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 K8s Job ↗).
Backoff Limit	Установите максимальное количество попыток повторного запуска перед пометкой Job как неудачного.

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 Pod'ы, 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 выполнений
  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 job с помощью**

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

- **Отслеживать жизненный цикл job с помощью**

```
kubectl get jobs
```

- **Просматривать детали выполнения с помощью**

```
kubectl describe job/<job-name>
```

- **Просматривать логи Pod с помощью**

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

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 запуска chart, называемый *release*.
Каждый раз при создании chart, обновлении или откате release создаётся инкрементальная ревизия.

1.1. Основные возможности

Helm предоставляет возможность:

- Искать большое количество charts в репозиториях charts
- Модифицировать существующие charts
- Создавать собственные charts с использованием ресурсов Kubernetes
- Упаковывать приложения и делиться ими в виде charts

1.2. Каталог

Каталог построен на основе Helm и представляет собой комплексную платформу управления распространением Chart, расширяющую ограничения инструмента Helm CLI. Платформа позволяет разработчикам удобнее управлять, развертывать и использовать charts через удобный интерфейс.

Определения терминов

Термин	Определение	Примечания
Application Catalog	Универсальная платформа управления Helm Charts	
Helm Charts	Формат упаковки приложений	
HelmRequest	CRD. Определяет конфигурацию, необходимую для развертывания Helm Chart	Template Application
ChartRepo	CRD. Соответствует репозиторию Helm charts	Template Repository
Chart	CRD. Соответствует Helm Charts	Template

1.3 Понимание HelmRequest

В Alauda Container Platform развертывания Helm в основном управляются через кастомный ресурс **HelmRequest**. Такой подход расширяет стандартный функционал Helm и интегрирует его непосредственно 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 развертываниям 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 отметить HelmRequest для развертывания как Application, добавив следующую аннотацию:

```
alauda.io/create-app: "true"
```

При включении этой функции UI платформы отображает 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 как простого pod, так и 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 требования

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 в репозиторий шаблонов типа Chart или OCI Chart с правами **Management**. Владельцы проектов, назначенные этому **Template Repository**, могут напрямую использовать шаблон.
Template Directory	При выборе типа репозитория шаблонов OCI Chart необходимо выбрать или вручную ввести директорию для хранения Helm Chart. **Примечание**: При ручном вводе новой директории платформа создаст эту директорию в репозитории шаблонов, но существует риск ошибки 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

Pod

Введение

Введение

Параметры Pod

Параметры Pod

Удаление Pods

Удаление Pods

Сценарии использования

Процедура

Контейнер

Введение

Debug Container (Alpha)

Принцип реализации

Примечания

Сценарии использования

Процедура

Вход в контейнер через EXEC

Вход в контейнер через приложения

Вход в контейнер через Pod

Menu

Введение

Обратитесь к официальной документации на сайте Kubernetes: Pod ↗

Menu

Параметры Pod

Интерфейс платформы предоставляет различную информацию о pod для быстрого ознакомления. Ниже приведены пояснения к некоторым параметрам.

Parameter	Description
Resource Requests/Limits	Эффективные значения запросов и лимитов ресурсов (CPU, память) для pod. Метод расчёта значений запросов и лимитов одинаков; в этом документе приведён пример с использованием значений лимитов, а конкретные правила и алгоритмы следующие: • Если pod содержит только **бизнес-контейнеры** (containers), значение лимита **CPU/памяти** равно сумме значений лимитов **CPU/памяти** всех контейнеров внутри pod. **Например**: если pod включает два бизнес-контейнера с лимитами **CPU/памяти** 100m/100Mi и 50m/200Mi, то значение лимита **CPU/памяти** для pod будет 150m/300Mi. • Если pod содержит как **init-контейнеры** (initContainers), так и **бизнес-контейнеры**, то шаги расчёта значений лимитов **CPU/памяти** для pod следующие: 1. Определить максимальное значение лимита **CPU/памяти** среди всех init-контейнеров. 2. Вычислить сумму значений лимитов **CPU/памяти** всех бизнес-контейнеров. 3. Сравнить результаты и взять максимальные значения **CPU** и **памяти** между init-контейнерами и бизнес-контейнерами в качестве значений лимитов **CPU/памяти** для pod. **Пример расчёта**: если pod содержит два init-контейнера с

Parameter	Description
	лимитами **CPU/памяти** 100m/200Mi и 200m/100Mi, максимальное значение лимита для init-контейнеров будет 200m/200Mi. При этом, если pod также содержит два бизнес-контейнера с лимитами **CPU/памяти** 100m/100Mi и 50m/200Mi, суммарное значение лимита для бизнес-контейнеров будет 150m/300Mi. Следовательно, комплексное значение лимита **CPU/памяти** для pod составит 200m/300Mi.
Source	Вычислительный компонент, к которому принадлежит pod.
Restart Count	Количество перезапусков при ненормальном состоянии pod.
Node	Имя узла, на котором расположен pod.
Service Account	Service Account — это учётная запись, которая позволяет процессам и сервисам в Pod обращаться к Kubernetes APIServer, предоставляя идентификацию для этих процессов и сервисов. Поле **Service Account** отображается только в том случае, если текущий вошедший пользователь обладает ролью администратора платформы или аудитора платформы, и при этом доступен просмотр YAML-файла Service Account.

Удаление Pods

Удаление pods может повлиять на работу вычислительных компонентов; пожалуйста, действуйте осторожно.

Содержание

Сценарии использования

- Быстрое восстановление pods до желаемого состояния: если pods остаётся в состоянии, которое влияет на бизнес-процессы, например `Pending` или `CrashLoopBackOff`, ручное удаление pods после устранения сообщения об ошибке поможет ему быстро вернуться в желаемое состояние, например `Running`. В этом случае удалённые pods будут восстановлены на текущем узле или переназначены.
- Очистка ресурсов для управления операциями: некоторые pods достигают определённой стадии, после которой они больше не изменяются, и такие группы часто накапливаются в большом количестве, усложняя управление другими pods. Pods, подлежащие очистке, могут включать те, что находятся в статусе `Evicted` из-за недостатка ресурсов узла, или в статусе `Completed`, вызванном повторяющимися запланированными задачами. В этом случае удалённые pods больше не будут существовать.

  Примечание: для запланированных задач, если необходимо проверять логи каждого выполнения задачи, не рекомендуется удалять соответствующие pods со статусом `Completed`.

Процедура

1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Workloads** > **Pods**.
3. (Удаление по одному) Нажмите ⋮ справа от pods, которые нужно удалить > **Delete**, и подтвердите.
4. (Массовое удаление) Выберите pods для удаления, нажмите **Delete** над списком и подтвердите.

Menu

Контейнер

Введение

Debug Container (Alpha)

Принцип реализации

Примечания

Сценарии использования

Процедура

Вход в контейнер через EXEC

Вход в контейнер через приложения

Вход в контейнер через Pod

Menu

Введение

Обратитесь к официальной документации сайта Kubernetes: Containers↗.

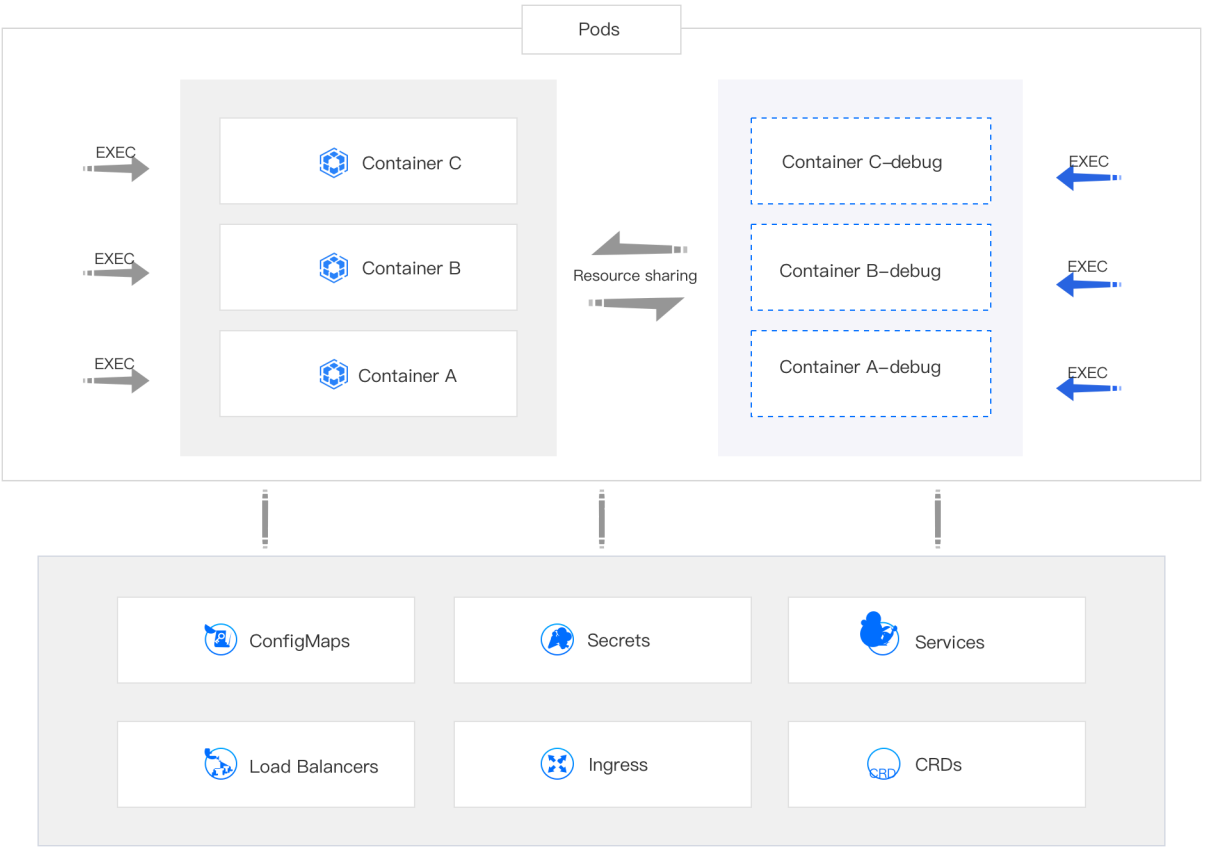
Debug Container (Alpha)

Функция Debug предоставляет необходимые инструменты для отладки запущенных контейнеров, включая системные, сетевые и дисковые утилиты.

Содержание

Принцип реализации

Функция Debug реализована через Ephemeral Containers. Ephemeral Container — это тип контейнера, который разделяет ресурсы с бизнес-контейнерами. Вы можете добавить Ephemeral Container (например, *Container A-debug*) в pod и использовать инструменты отладки внутри этого контейнера. Результаты отладки будут напрямую применяться к бизнес-контейнеру (например, *Container A*).



Примечания

- Нельзя добавить Ephemeral Container напрямую через обновление конфигурации pod; убедитесь, что Ephemeral Container включён через функцию Debug.
- Ephemeral Containers, включённые функцией Debug, не имеют гарантий ресурсов или планирования и не будут перезапускаться автоматически. Пожалуйста, избегайте запуска в них бизнес-приложений, кроме целей отладки.
- Используйте функцию Debug с осторожностью, если ресурсы на узле, где расположен pod, близки к исчерпанию, так как это может привести к эвакуации pod.

Сценарии использования

Хотя вы также можете войти в контейнер и отлаживать с помощью функции EXEC, многие образы контейнеров не включают необходимые инструменты отладки (например, bash, net-tools и др.) для уменьшения размера образа. В отличие от этого,

функция Debug, поставляемая с предустановленными инструментами отладки, более подходит для следующих случаев.

- Диагностика неисправностей: если в бизнес-контейнере возникает проблема, помимо проверки событий и логов, может потребоваться более детальное устранение неполадок и решение внутри контейнера.
- Настройка конфигурации: если в текущем бизнес-решении есть недостатки, вы можете выполнить настройку конфигурации бизнес-компонентов внутри контейнера, чтобы разработать новую схему конфигурации, которая поможет бизнесу работать эффективнее.

Процедура

1. Войдите в **Container Platform**.
2. В левой навигационной панели нажмите **Workloads** > **Pods**.
3. Найдите нужный pod и нажмите ⋮ > **Debug**.
4. Выберите контейнер, который хотите отлаживать.
5. (Опционально) Если интерфейс запросит **инициализацию**, нажмите **Initialize**.

   Примечание: После инициализации функции Debug, пока pod не будет пересоздан, вы можете напрямую войти в Ephemeral Container (например, *Container A-debug*) для отладки.
6. Дождитесь готовности окна отладки и начните отладку.

   Совет: Нажмите на запрос команд в правом верхнем углу, чтобы просмотреть распространённые инструменты и их использование.
7. По окончании закройте окно отладки.

Вход в контейнер через EXEC

Содержание

Вход в контейнер через приложения

Вы можете войти во внутренний экземпляр контейнера с помощью команды `kubectl exec`, что позволяет выполнять операции в командной строке в окне веб-консоли. Кроме того, вы можете легко загружать и скачивать файлы внутри контейнера с помощью функции передачи файлов.

Требования

- Контейнер должен работать корректно.
- При использовании функции передачи файлов в контейнере должен быть доступен инструмент `tar`, а операционная система контейнера не должна быть Windows.

Порядок действий

1. Перейдите в **Container Platform**.

2. В левой навигационной панели нажмите **Application** > **Applications**.
3. Нажмите на ***Application Name***.
4. Найдите workload и нажмите **EXEC** > ***Pod Name***.
5. Введите команду, которую хотите выполнить.
6. Нажмите **OK**, чтобы войти в окно веб-консоли и выполнить операции в командной строке.
7. Нажмите **File Transfer**. Введите **Upload Path** для загрузки файлов для тестирования в контейнер; или введите **Download Path** для скачивания логов и других файлов из контейнера на локальную машину для анализа.

Вход в контейнер через Pod

Вы можете войти во внутренний экземпляр контейнера с помощью команды `kubectl exec`, что позволяет выполнять операции в командной строке в окне веб-консоли. Кроме того, вы можете легко загружать и скачивать файлы внутри контейнера с помощью функции передачи файлов.

Требования

- Контейнер должен работать корректно.
- При использовании функции передачи файлов в контейнере должен быть доступен инструмент `tar`, а операционная система контейнера не должна быть Windows.

Порядок действий

1. В левой навигационной панели нажмите **Workloads** > **Pods**.
2. Нажмите ⋮ > **EXEC** > ***Container Name***.
3. Введите команду, которую хотите выполнить.

4. Нажмите **OK**, чтобы войти в окно веб-консоли и выполнить операции в командной строке.

5. Нажмите **File Transfer**. Введите **Upload Path** для загрузки файлов для тестирования в контейнер; или введите **Download Path** для скачивания логов и других файлов из контейнера на локальную машину для анализа.

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

Реестр

Введение

Введение

Принципы и изоляция 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

Common CLI Command Operations

Вход в реестр

Добавление разрешений на пространство имён для пользователей

Добавление разрешений на пространство имён для сервисного аккаунта

Загрузка образов

Отправка образов

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 более высокую эффективность внутреннего доступа по сравнению с внешним, независимо развернутым репозиторием образов.

Содержание

Принципы и изоляция namespace

Встроенный репозиторий образов Alauda Container Platform, как один из ключевых компонентов платформы, работает внутри кластера в режиме высокой доступности и

использует возможности постоянного хранения, предоставляемые платформой, чтобы гарантировать безопасность и надёжность данных образов.

Одним из основных концептов его проектирования является логическая изоляция и управление на основе Namespace. 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 только явно не предоставлено разрешение.

Аутентификация и авторизация

Механизм аутентификации и авторизации Alauda Container Platform Registry глубоко интегрирован с системой аутентификации и авторизации уровня платформы ACP, обеспечивая контроль доступа с 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 паролей для Registry. Они аутентифицируются через стандартные механизмы аутентификации платформы (например, с использованием API токенов, предоставляемых платформой, интегрированных корпоративных систем идентификации и т.п.). При доступе к Alauda Container Platform Registry через CLI или другие инструменты обычно используется существующая сессия входа на платформу или токены ServiceAccount для прозрачной аутентификации.

Авторизация

Контроль авторизации реализован на уровне namespace. Права Pull или Push для репозитория образов в Alauda Container Platform Registry зависят от роли и разрешений, которые пользователь или ServiceAccount имеют в соответствующем namespace платформы.

- **Как правило**, владельцу или роли разработчика namespace автоматически предоставляются права Push и Pull к репозиториям образов в этом namespace.
- **Пользователи из других namespace или пользователи, желающие выполнять pull образов между namespace**, должны получить явное разрешение от администратора целевого namespace (например, привязка роли, позволяющей выполнять pull образов через RBAC), прежде чем они смогут 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

- **Лёгкое развертывание:** Реализация упрощённых решений реестра в средах с низкой нагрузкой для ускорения доставки приложений.
- **Edge Computing:** Обеспечение автономного управления для edge-кластеров с выделенными реестрами.

- **Оптимизация ресурсов:** Демонстрация возможностей полного рабочего процесса через интегрированные решения Source to Image (S2I) при недостаточном использовании инфраструктуры.

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.
- **Производственных развертываний**, требующих корпоративного хранилища (NAS, AWS S3, Ceph и др.).
- Сред, где необходим **тонкий контроль** над TLS, ingress.
- **Полной настройки YAML** для сложных конфигураций.

Предварительные требования

- **Установить** плагин кластера **Alauda Container Platform Registry** в целевой кластер.
- **Доступ** к целевому Kubernetes кластеру с настроенным kubectl.
- **Права администратора кластера** для создания ресурсов с областью действия кластера.
- Получить зарегистрированный **домен** (например, registry.yourcompany.com) Create a Domain
- Обеспечить действующее **NAS-хранилище** (например, NFS, GlusterFS и др.).
- (Опционально) Обеспечить действующее **S3-хранилище** (например, AWS S3, Ceph и др.). Если S3-хранилище отсутствует, разверните MinIO (встроенный S3) в кластере Deploy MinIO.

Установка Alauda Container Platform Registry через YAML

Процедура

1. **Создайте YAML-файл конфигурации** с именем registry-plugin.yaml по следующему шаблону:

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ClusterPluginInstance
metadata:
  annotations:
    cpaas.io/display-name: 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
      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-бакета
      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>  # Endpoint 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:
    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** с учетными данными 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 к вашему кластеру:

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

Справочник по конфигурации

Обязательные поля

Параметр	Описание	Пример значения
`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`
`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-бакета для хранения	`prod-image-store`

Параметр	Описание	Пример значения
	образов	
`spec.config.s3storage.region`	Регион AWS для S3 хранилища	`us-west-1`
`spec.config.s3storage.regionEndpoint`	URL endpoint сервиса S3	`https://s3.amazonaws.com`
`spec.config.s3storage.secretName`	Секрет, содержащий учетные данные 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 в глобальном кластере::

```
# <CLUSTER-NAME> - кластер, в котором 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 в глобальном кластере:

```
# <CLUSTER-NAME> - кластер, в котором 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, впервые работающих с системой**, предпочитающих пошаговый визуальный интерфейс.
- **Быстрой настройки proof-of-concept** в непроизводственных средах.
- Команд с **ограниченными знаниями Kubernetes**, которым нужен упрощённый процесс развертывания.
- Сценариев, требующих **минимальной кастомизации** (например, стандартные настройки хранилища).
- **Базовых сетевых конфигураций** без специфичных правил ingress.
- Конфигураций **StorageClass** для обеспечения высокой доступности.

Не рекомендуется для:

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

- Сетевых конфигураций с необходимостью специфичных правил ingress.

Предварительные требования

- **Установите** плагин кластера **Alauda Container Platform Registry** в целевой кластер с помощью механизма Cluster Plugin.

Установка плагина кластера Alauda Container Platform Registry через веб-консоль

Процедура

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace** > **Cluster Plugins**, чтобы открыть страницу списка **Cluster Plugins**.
3. Найдите плагин кластера **Alauda Container Platform Registry**, нажмите **Install**, затем перейдите на страницу установки.
4. Настройте параметры согласно следующим спецификациям и нажмите **Install** для завершения развертывания.

Описание параметров:

Параметр	Описание
Expose Service	При включении администраторы смогут управлять репозиторием образов извне по адресу доступа. Это создает значительные риски безопасности, поэтому включать следует с большой осторожностью.
Enable IPv6	Включите эту опцию, если в кластере используется сетевая конфигурация IPv6 single-stack.

Параметр	Описание
NodePort	При включённом Expose Service настройте NodePort для обеспечения внешнего доступа к Registry через этот порт.
Storage Type	Выберите тип хранилища. Поддерживаемые типы: LocalVolume и StorageClass.
Nodes	Выберите узел для запуска сервиса Registry для хранения и распространения образов. (Доступно только при выборе Storage Type = LocalVolume)
StorageClass	Выберите StorageClass. При количестве реплик больше 1 выберите хранилище с возможностью RWX (ReadWriteMany) (например, File Storage) для обеспечения высокой доступности. (Доступно только при StorageClass)
Storage Size	Объем хранилища, выделенный для Registry (единица измерения: Gi).
Replicas	Настройте количество реплик для Pod Registry: • **LocalVolume**: по умолчанию 1 (фиксировано) • **StorageClass**: по умолчанию 3 (можно изменить)
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

Common CLI Command Operations

Вход в реестр

Добавление разрешений на пространство имён для пользователей

Добавление разрешений на пространство имён для сервисного аккаунта

Загрузка образов

Отправка образов

Using Alauda Container Platform Registry in Kubernetes Clusters

Registry Access Guidelines

Deploy Sample Application

Cross-Namespace Access

Best Practices

Verification Checklist

Troubleshooting

Common CLI Command Operations

Платформа Alauda Container Platform предоставляет инструменты командной строки для взаимодействия 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 для получения плагина kubectl-acp и убедитесь, что он корректно 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

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 права на pull в целевом пространстве имён, а затем загружает образ из реестра.

Отправка образов

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

Сначала необходимо пометить (tag) локальный образ адресом и форматом пространства имён целевого реестра с помощью стандартного инструмента командной строки для контейнеров, например docker.

```
# Пометить образ 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 конкретное пространство имён реестра Alauda Container Platform.

```
# Если в вашем удалённом репозитории 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 права на push в пространстве имён my-ns, а затем загружает локально помеченный образ в реестр.

Using Alauda Container Platform Registry in Kubernetes Clusters

Реестр Alauda Container Platform (ACP) обеспечивает безопасное управление образами контейнеров для рабочих нагрузок Kubernetes.

Содержание

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, CI/CD систем)
- 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 разрешений.

Example Role Binding

```
# Доступ к образам из другого пространства имён (требуются разрешения)
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 улучшения безопасности и управления образами.
  - Используйте пути образов на основе пространства имён: `internal-docker-registry.cpaas-system.svc/<namespace>/<image>:<tag>`.
- **Контроль доступа**: Используйте role binding 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 binding в целевом пространстве имён.
- **Проблемы с сетью**: Проверьте сетевые политики и конфигурации сервисов для обеспечения подключения к внутреннему реестру.
- **Сбои DNS**: Проверьте содержимое файла `/etc/hosts` на узле, убедитесь, что разрешение DNS для `internal-docker-registry.cpaas-system.svc` настроено корректно.
  - Проверьте конфигурацию /etc/hosts на узле 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

Введение

Введение

Концепция Source to Image

Основные возможности

Основные преимущества

Сценарии применения

Ограничения использования

Установка

Установка сборок Alauda Container Platform

Предварительные требования

Процедура

Архитектура

Архитектура

Руководства

Управление приложениями, созданными из кода

Основные возможности

Преимущества

Требования

Процедура

Связанные операции

Как сделать

Создание приложения из кода

Предварительные требования

Процедура

Введение

Alauda Container Platform Builds — это облачный контейнерный инструмент, предоставляемый Alauda Container Platform , который объединяет возможности Source to Image (S2I) с автоматизированными конвейерами. Он ускоряет переход предприятий к облачным нативным технологиям, обеспечивая полностью автоматизированные CI/CD конвейеры, поддерживающие несколько языков программирования, включая Java, Go, Python и Node.js. Кроме того, Alauda Container Platform Builds предлагает визуальное управление релизами и бесшовную интеграцию с Kubernetes-native инструментами, такими как Helm и GitOps, обеспечивая эффективное управление жизненным циклом приложений от разработки до производства.

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 конечном итоге генерируя запускаемый контейнерный образ. Это позволяет разработчикам больше сосредоточиться на разработке бизнес-логики, не беспокоясь о деталях контейнеризации.

Основные возможности

Alauda Container Platform Builds обеспечивает полный стек облачного нативного рабочего процесса от кода до приложения, поддерживая сборки на нескольких языках и визуальное управление релизами. Он использует возможности Kubernetes-native для преобразования исходного кода в запускаемые контейнерные образы, обеспечивая бесшовную интеграцию в комплексную облачную платформу.

- **Сборка на нескольких языках**: поддержка сборки приложений на различных языках программирования, таких как Java, Go, Python и Node.js, что удовлетворяет разнообразные потребности разработки.
- **Визуальный интерфейс**: предоставляет интуитивно понятный интерфейс, позволяющий легко создавать, настраивать и управлять задачами сборки без глубоких технических знаний.
- **Полное управление жизненным циклом**: охватывает весь жизненный цикл от коммита кода до развертывания приложения, автоматизируя сборку, развертывание и операционное управление.
- **Глубокая интеграция**: бесшовно интегрируется с вашим продуктом Container Platform, обеспечивая непрерывный опыт разработки.
- **Высокая расширяемость**: поддерживает пользовательские плагины и расширения для удовлетворения ваших специфических требований.

Основные преимущества

- **Ускоренная разработка**: оптимизирует процесс сборки, ускоряя доставку приложений.
- **Повышенная гибкость**: поддержка сборки на нескольких языках программирования.
- **Повышенная эффективность**: автоматизирует процессы сборки и развертывания, снижая необходимость ручного вмешательства.
- **Повышенная надежность**: предоставляет подробные логи сборки и визуальный мониторинг для облегчения устранения неполадок.

Сценарии применения

Основные сценарии применения S2I следующие:

- Веб-приложения

  S2I поддерживает различные языки программирования, такие как Java, Go, Python и Node.js. Используя возможности управления приложениями Alauda Container Platform , он позволяет быстро создавать и развертывать веб-приложения, просто указав URL репозитория кода.

- CI/CD

  S2I бесшовно интегрируется с DevOps конвейерами, 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: Tekton Operator теперь доступен в OperatorHub кластера.

Menu

Установка

Установка сборок Alauda Container Platform

Содержание

Предварительные требования

> Alauda Container Platform Builds — это контейнерный инструмент, предлагаемый Alauda Container Platform , который объединяет процесс сборки (включая Source to Image) и создание приложений.

1. Скачайте последнюю версию пакета **Alauda Container Platform Builds**, соответствующую вашей платформе. Если оператор **Tekton** ещё не установлен в Kubernetes-кластере, рекомендуется скачать его вместе с пакетом.
2. Используйте CLI-инструмент `violet` для загрузки пакетов **Alauda Container Platform Builds** и **Tekton** в целевой кластер. Подробные инструкции по использованию `violet` доступны в разделе CLI.

Процедура

Установка оператора Alauda Container Platform Builds

1. Выполните вход и перейдите на страницу **Platform Management**.
2. Нажмите **Marketplace** > **OperatorHub**.
3. Найдите оператора **Alauda Container Platform Builds**, нажмите **Install** и перейдите на страницу **Install**.

Параметры конфигурации:

Параметр	Рекомендуемая конфигурация
Channel	`Alpha`: По умолчанию выбран канал **alpha**.
Version	Выберите последнюю версию.
Installation Mode	`Cluster`: Один оператор используется для всех пространств имён в кластере для создания и управления инстансами, что снижает использование 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**.

Установка инстанса Shipyard

1. Нажмите **Marketplace** > **OperatorHub**.
2. Найдите установленного оператора **Alauda Container Platform Builds**, перейдите в раздел **All Instances**.

3. Нажмите кнопку **Create Instance**, затем выберите карточку **Shipyard** в области ресурсов.

4. На странице настройки параметров инстанса можно использовать конфигурацию по умолчанию, если нет специальных требований.

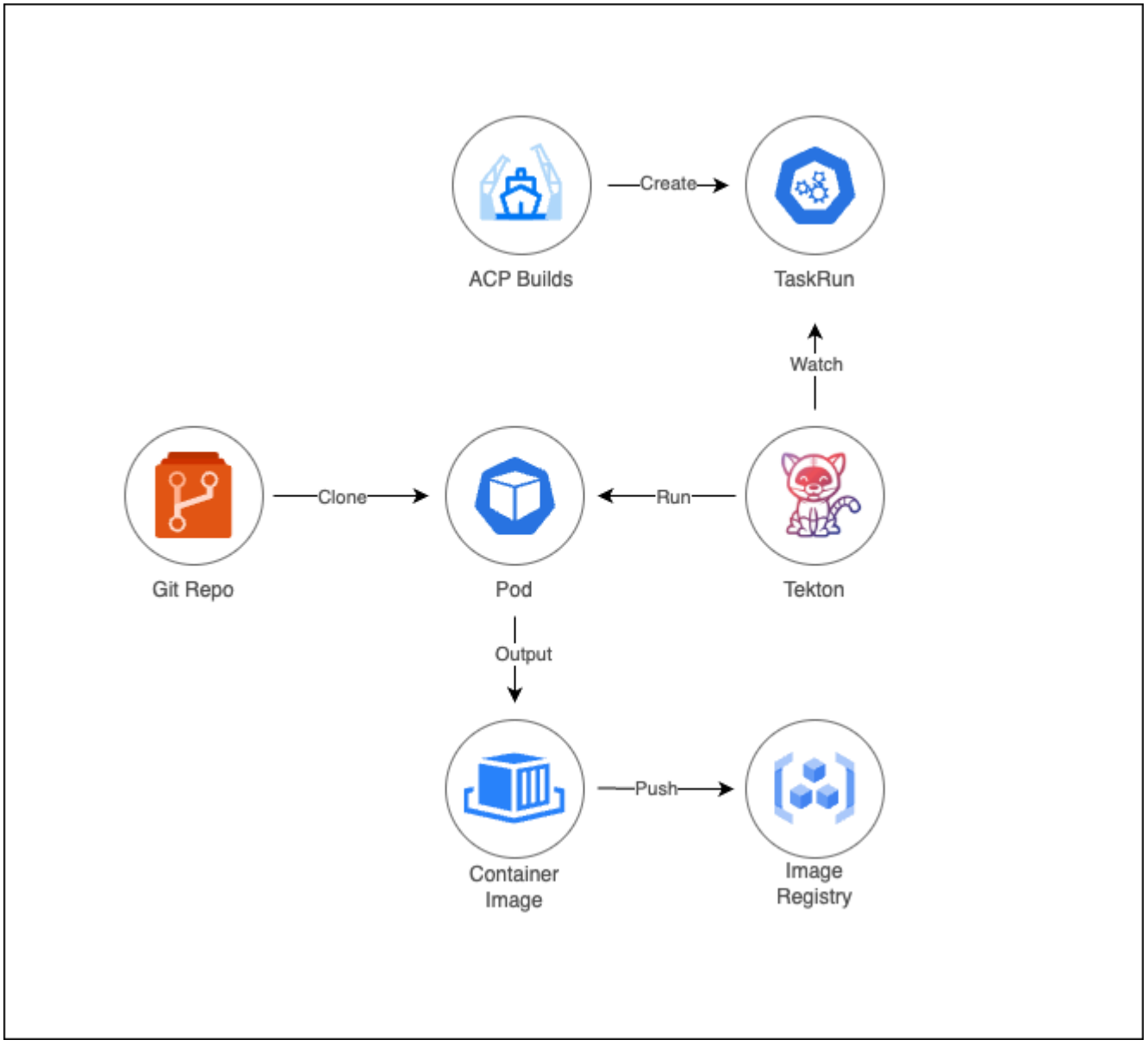
5. Нажмите **Create**.

Проверка

- После успешного создания инстанса подождите примерно 20 минут, затем перейдите в **Container Platform** > **Applications** > **Applications** и нажмите **Create**.
- Вы должны увидеть пункт **Create from Code**. Это означает, что установка Alauda Container Platform Builds прошла успешно, и вы можете начать работу с S2I, следуя инструкции Creating an application from Code.

Menu

Архитектура



Возможность Source to Image (S2I) реализована через оператор **Alauda Container Platform Builds**, обеспечивающий автоматизированную сборку контейнерных образов из исходного кода репозитория Git и последующую отправку в указанный реестр образов. Основные компоненты включают:

- Оператор **Alauda Container Platform Builds**: Управляет полным циклом сборки и оркестрирует конвейеры Tekton.
- Конвейеры **Tekton**: Выполняют рабочие процессы S2I с помощью Kubernetes-нативных ресурсов `TaskRun`.

Menu

Руководства

Управление приложениями, созданными из кода

Основные возможности

Преимущества

Требования

Процедура

Связанные операции

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 управление эксплуатацией, облегчая локализацию проблем, анализ и устранение неполадок.

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, интегрированный с платформой и уже выделенный для текущего проекта; платформа поддерживает GitLab, GitHub и Bitbucket. • **Input**: Используйте URL репозитория кода, не интегрированного с платформой.

Имя интегрированного проекта	Имя проекта интеграционного инструмента, назначенное или связанное 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, но при необходимости можно выбрать и другие версии.
Context dir	Необязательный каталог исходного кода, используемый как контекстный каталог для сборки.

	Secret	При использовании входного репозитория кода можно при необходимости добавить секрет аутентификации.
	Builder Image	• Образ, включающий конкретные среды выполнения языков программирования, библиотеки зависимостей и скрипты S2I. Его основная задача — преобразование исходного кода в исполняемые образы приложений. • Поддерживаемые builder images включают: Golang, Java, Node.js и Python.
	Версия	Выберите версию среды выполнения, совместимую с вашим исходным кодом, чтобы обеспечить корректное выполнение приложения.
Сборка	**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 Source-to-Image (S2I) сборки. Подробнее 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 соответствующую 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: загрузку исходного кода из репозитория.

		Вызов указанного builder image. Выполнение процесса сборки.
	URL образа	После завершения сборки укажите адрес целевого репозитория образов для приложения.
Приложение	-	Заполните конфигурацию приложения по необходимости. Для подробностей см. описание 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**: См. описание параметров в документации CreatingIngress.
Метки и аннотации	-	Заполните соответствующие метки и аннотации по необходимости.

5. После заполнения параметров нажмите **Create**.

6. Вы можете просмотреть соответствующее развертывание на странице **Details**.

Связанные операции

Сборка

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 данной функции убедитесь, что:

- Установлен Alauda Container Platform Builds
- На платформе доступен репозиторий образов. Если нет, обратитесь к администратору для установки Alauda Container Platform Registry

Процедура

1. В **Container Platform** перейдите в **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

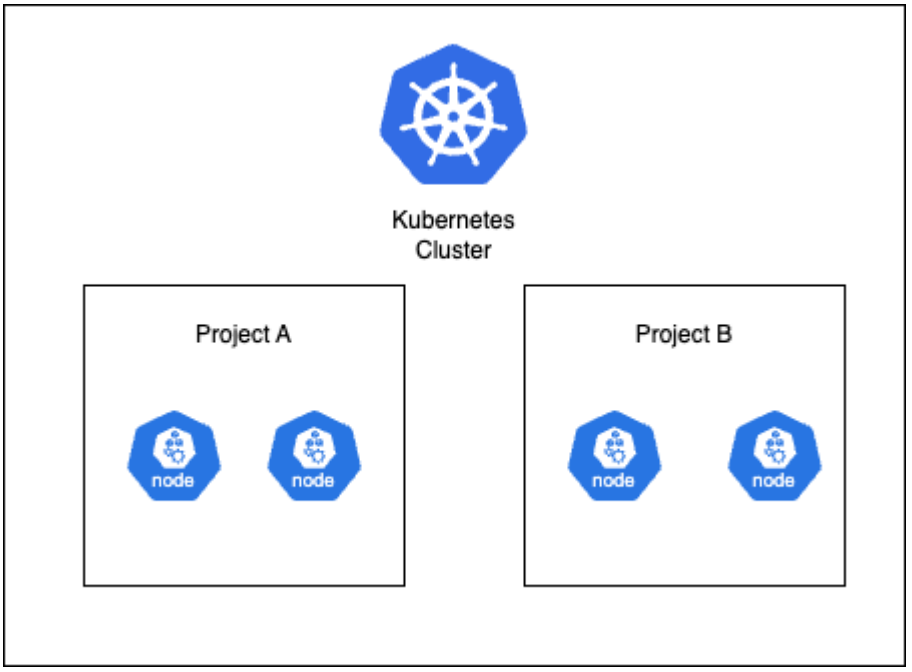
Удобное выделение узлов проектам в эксклюзивном или совместном режиме, предотвращая конкуренцию за ресурсы между проектами.

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 наименьшее значение для каждого элемента квоты среди всех ResourceQuota и объединит их, в итоге создав один ResourceQuota с именем `default`.

Пример:

Namespace `to-import`, который нужно импортировать, содержит следующие ресурсы `resourcequota`:

```
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: a
  namespace: to-import
spec:
  hard:
    requests.cpu: "1"
    requests.memory: "500Mi"
    limits.cpu: "3"
    limits.memory: "1Gi"
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: b
  namespace: to-import
spec:
  hard:
    requests.cpu: "2"
    requests.memory: "300Mi"
    limits.cpu: "2"
    limits.memory: "2Gi"
```

После импорта namespace `to-import` в нем будет создан следующий ResourceQuota с именем `default`:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: default
  namespace: to-import
spec:
  hard:
    requests.cpu: "1"
    requests.memory: "300Mi"
    limits.cpu: "2"
    limits.memory: "1Gi"
```

Для каждого ResourceQuota квоты ресурсов — это минимальное значение между `a` и `b`.

Когда в namespace существует несколько ResourceQuota, Kubernetes проверяет каждую ResourceQuota независимо. Поэтому после импорта namespace рекомендуется удалить все ResourceQuota, кроме `default`. Это помогает избежать сложностей в расчетах квот из-за нескольких ResourceQuota, что может легко привести к ошибкам.

Почему при импорте namespace не должно быть нескольких LimitRange или LimitRange с именем, отличным от `default`?

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.

Платформа создает LimitRange с именем `default` при создании namespace. Поэтому перед импортом namespace в нем должен существовать только один LimitRange с именем `default`.