Menu

Хранилище

Введение

Введение

Основные понятия

Основные понятия

Persistent Volume (PV)

Persistent Volume Claim (PVC)

Generic Ephemeral Volumes

emptyDir

hostPath

ConfigMap

Secret

StorageClass

Container Storage Interface (CSI)

Persistent Volume

Динамические Persistent Volumes и статические Persistent Volumes

Жизненный цикл Persistent Volumes

Режимы доступа и режимы томов

Режимы доступа в Kubernetes

Режимы томов в Kubernetes

Особенности хранения: снимки и расширение

Заключение

Руководства

Создание Storage Class типа CephFS File Storage

Развертывание Volume Plugin

Создание Storage Class

Создание класса блочного хранилища CephRBD

Развертывание плагина тома

Создание класса хранилища

Создание локального Storage Class TopoLVM

Общая информация

Развертывание Volume Plugin

Создание Storage Class

Последующие действия

Создание общего класса хранения NFS

Предварительные требования

Развертывание плагина Alauda Container Platform NFS CSI

Создание общего класса хранения NFS

Развертывание компонента Volume Snapshot

Развертывание через веб-консоль

Развертывание через YAML

Создание PV

Предварительные требования

Пример PersistentVolume

Создание PV через веб-консоль

Создание PV через CLI

Связанные операции

Дополнительные ресурсы

Создание PVC

Предварительные требования

Пример PersistentVolumeClaim:

Создание Persistent Volume Claim с помощью веб-консоли

Создание Persistent Volume Claim с помощью CLI

Операции

Расширение ёмкости PersistentVolumeClaim с помощью веб-консоли

Расширение ёмкости Persistent Volume Claim с помощью CLI

Дополнительные ресурсы

Использование снимков томов

Как сделать

Generic ephemeral volumes

Использование emptyDir

Настройка постоянного хранилища с использованием NFS

Руководство по аннотированию возможностей стороннего хранилища

Устранение неполадок

Восстановление после ошибки расширения PVC

Menu

Введение

Kubernetes предлагает гибкий и масштабируемый механизм хранения для управления сохранением данных в контейнеризованных средах. Абстрагируя ресурсы хранения, такие как Volumes, PersistentVolumes и PersistentVolumeClaims, Kubernetes отделяет приложения от базовых систем хранения, обеспечивая динамическое выделение, автоматическое монтирование и сохранение данных между узлами.

Ключевые возможности включают поддержку множества систем хранения (например, локальные диски, NFS, облачные сервисы хранения), динамическое выделение, контроль режимов доступа (например, права на чтение/запись) и управление жизненным циклом — что удовлетворяет потребности в хранении для stateful-приложений. Для корпоративных нагрузок, требующих высокой доступности, сохранения данных и изоляции многопользовательской среды, хранение в Kubernetes является важной базовой функцией.

Хранение в Kubernetes разработано для разработчиков, инженеров эксплуатации и платформенных команд, помогая им эффективно и безопасно управлять данными в контейнеризованных нагрузках.

Menu

Основные понятия

Основные понятия

Persistent Volume (PV)

Persistent Volume Claim (PVC)

Generic Ephemeral Volumes

emptyDir

hostPath

ConfigMap

Secret

StorageClass

Container Storage Interface (CSI)

Persistent Volume

Динамические Persistent Volumes и статические Persistent Volumes

Жизненный цикл Persistent Volumes

Режимы доступа и режимы томов

Режимы доступа в Kubernetes

Режимы томов в Kubernetes

Особенности хранения: снимки и расширение

Заключение

Основные понятия

Хранение данных в Kubernetes основано на трех ключевых понятиях: **PersistentVolume (PV)**, **PersistentVolumeClaim (PVC)** и **StorageClass**. Они определяют, как запрашивается, выделяется и настраивается хранилище внутри кластера. В основе часто лежат драйверы **CSI** (Container Storage Interface), которые отвечают за фактическое предоставление и подключение хранилища. Давайте кратко рассмотрим каждый компонент, а затем выделим роль CSI-драйвера.

Содержание

Persistent Volume (PV)

PersistentVolume (PV) — это часть хранилища в кластере, которая была выделена (либо статически администратором, либо динамически через StorageClass). Он представляет собой базовое хранилище — например, диск у облачного провайдера или

сетевую файловую систему — и рассматривается как ресурс в кластере, аналогично узлу.

Persistent Volume Claim (PVC)

PersistentVolumeClaim (PVC) — это запрос на хранилище. Пользователи определяют, сколько хранилища им нужно и режим доступа (например, чтение-запись). Если подходящий PV доступен или может быть динамически создан (через StorageClass), PVC связывается с этим PV. После связывания Pod'ы могут ссылаться на PVC для сохранения или совместного использования данных.

Generic Ephemeral Volumes

Generic Ephemeral Volumes для Kubernetes — это функция, введённая в Kubernetes, которая позволяет использовать CSI-управляемые `временные` тома в течение жизненного цикла Pod, аналогично emptyDir, но более мощная и позволяющая монтировать любой тип CSI тома (с поддержкой снимков, масштабирования и т. д.).

Для дополнительной информации смотрите Generic ephemeral volumes

---

emptyDir

1. emptyDir — это временный том типа пустой директории.
2. Он создаётся при запуске Pod на узле, и хранилище располагается на локальной файловой системе этого узла (по умолчанию диск узла).
3. При удалении Pod данные в emptyDir также удаляются.

Для дополнительной информации смотрите Using an emptyDir

hostPath

В Kubernetes том hostPath — это специальный тип тома, который отображает файл или директорию с файловой системы хост-узла непосредственно в контейнер Pod.

- Позволяет Pod получить доступ к файлам или директориям на узле-хосте.
- Полезно для:
  - Доступа к ресурсам уровня хоста (например, сокет Docker)
  - Отладки
  - Использования уже существующих данных на узле

ConfigMap

ConfigMap в Kubernetes — это объект API, используемый для хранения неконфиденциальных данных конфигурации в виде пар ключ-значение. Он позволяет отделить конфигурацию от кода приложения, делая приложения более переносимыми и удобными в управлении.

Secret

В Kubernetes Secret — это объект API, который хранит конфиденциальные данные, такие как:

- пароли
- OAuth токены
- SSH ключи
- TLS сертификаты

- учётные данные баз данных

Secrets помогают защитить эти данные, избегая их прямого хранения в спецификациях Pod или образах контейнеров.

StorageClass

StorageClass описывает *как* тома должны динамически выделяться. Он сопоставляется с конкретным провиженером (часто CSI-драйвером) и может включать параметры, такие как уровни хранения, характеристики производительности или другие настройки бэкенда. Создавая несколько StorageClass, вы можете предложить разработчикам различные типы хранилища.

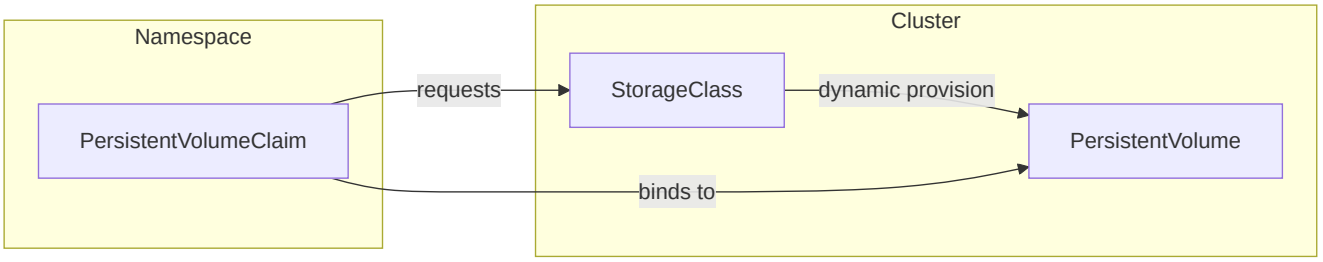

Диаграмма: Взаимосвязь между PVC, PV и StorageClass.

Container Storage Interface (CSI)

Container Storage Interface (CSI) — это стандартный API, который Kubernetes использует для интеграции с драйверами хранилища. Он позволяет сторонним провайдерам создавать плагины вне ядра Kubernetes, то есть вы можете устанавливать или обновлять драйвер хранилища без изменения самого Kubernetes.

CSI **драйвер** обычно состоит из двух компонентов:

1. **Компонент контроллера**: работает в кластере (часто как Deployment) и отвечает за операции высокого уровня, такие как **создание** или **удаление** томов. Для сетевого хранилища он также может управлять подключением и отключением томов к узлам.

2. **Компонент узла**: работает на каждом узле (часто как DaemonSet) и отвечает за **монтирование** и **отмонтирование** тома на конкретном узле. Он взаимодействует с kubelet, чтобы обеспечить доступность тома для Pod.

Когда пользователь создаёт PVC, ссылающийся на StorageClass с CSI-драйвером, драйвер отслеживает этот запрос и выделяет хранилище при необходимости динамического провиженинга. После создания хранилища драйвер уведомляет Kubernetes, который создаёт соответствующий PV и связывает его с PVC. Когда Pod использует этот PVC, компонент узла драйвера обрабатывает монтирование тома, делая хранилище доступным внутри контейнера.

Используя **PV**, **PVC**, **StorageClass** и **CSI**, Kubernetes предоставляет мощный декларативный подход к управлению хранилищем. Администраторы могут определить один или несколько StorageClass для представления различных бэкендов или уровней производительности, а разработчики просто запрашивают хранилище через PVC — не заботясь об инфраструктуре под капотом.

Persistent Volume

PersistentVolume (PV) представляет собой объект Kubernetes API, отображающий связь с томами бэкенд-хранилища в кластере Kubernetes. Это ресурс кластера, который создаётся и настраивается администраторами единообразно, отвечая за абстрагирование реальных ресурсов хранения и формирование инфраструктуры хранения кластера.

PersistentVolumes имеют жизненный цикл, независимый от Pod'ов, что позволяет обеспечивать постоянное хранение данных Pod'ов.

Администраторы могут вручную создавать статические PersistentVolumes или генерировать динамические PersistentVolumes на основе классов хранения. Если разработчикам необходимо получить ресурсы хранения для приложений, они могут запросить их через PersistentVolumeClaims (PVC), которые сопоставляются и связываются с подходящими PersistentVolumes.

Содержание

Динамические Persistent Volumes и статические Persistent Volumes

Платформа поддерживает управление двумя типами PersistentVolumes администраторами — динамическими и статическими Persistent Volumes.

- **Динамические Persistent Volumes**: Реализуются на основе классов хранения. Классы хранения создаются администраторами и представляют собой ресурс Kubernetes, описывающий категорию ресурсов хранения. Как только разработчик создаёт PersistentVolumeClaim, связанный с классом хранения, платформа динамически создаёт подходящий PersistentVolume в соответствии с параметрами, настроенными в PersistentVolumeClaim и классе хранения, связывая его с PersistentVolumeClaim для динамического выделения ресурсов хранения.

- **Статические Persistent Volumes**: Persistent Volumes, создаваемые вручную администратором. В настоящее время поддерживается создание статических Persistent Volumes типов **HostPath** или **NFS shared storage**. Когда разработчики создают PersistentVolumeClaim без использования класса хранения, платформа сопоставляет и связывает подходящий статический PersistentVolume в соответствии с параметрами, настроенными в PersistentVolumeClaim.

  - **HostPath**: Использует файловый каталог на хосте узла (локальное хранилище не поддерживается) в качестве бэкенд-хранилища, например: `/etc/kubernetes`. Обычно применяется только для тестовых сценариев в кластере с одним вычислительным узлом.

  - **NFS Shared Storage**: Относится к Network File System — распространённому типу бэкенд-хранилища для Persistent Volumes. Пользователи и программы могут обращаться к файлам на удалённых системах так, как если бы они были локальными.

Жизненный цикл Persistent Volumes

1. **Provisioning (создание)**: Администраторы вручную создают статические Persistent Volumes. После создания Persistent Volume переходит в состояние **Available**; альтернативно, платформа динамически создаёт подходящие Persistent Volumes на основе PersistentVolumeClaims, связанных с классами хранения.

2. **Binding (связывание)**: После того как статический Persistent Volume сопоставлен и связан с PersistentVolumeClaim, он переходит в состояние **Bound**; динамические Persistent Volumes создаются динамически на основе запросов, соответствующих

PersistentVolumeClaims, и также переходят в состояние **Bound** после успешного создания.

3. **Using (использование)**: Разработчики связывают PersistentVolumeClaims с экземплярами контейнеров вычислительных компонентов, используя ресурсы бэкенд-хранилища, отображённые Persistent Volumes.

4. **Releasing (освобождение)**: После удаления разработчиками PersistentVolumeClaim Persistent Volume освобождается.

5. **Reclaiming (восстановление)**: После освобождения Persistent Volume на нём выполняются операции восстановления в соответствии с параметрами политики восстановления Persistent Volume или класса хранения.

Режимы доступа и режимы томов

В Kubernetes PersistentVolumeClaims (PVC) и StorageClasses работают вместе для управления тем, как хранилище предоставляется и используется рабочими нагрузками. Два ключевых понятия в этой области — это **режимы доступа** и **режимы томов**. В этой статье рассматриваются эти понятия и подчеркивается, как различные системы хранения поддерживают их.

Содержание

Режимы доступа в Kubernetes

Режимы доступа определяют, как том может быть смонтирован и использован подами. Основные режимы доступа:

- **ReadWriteOnce (RWO)**: том может быть смонтирован в режиме чтения-записи одним узлом.
- **ReadOnlyMany (ROX)**: том может быть смонтирован в режиме только для чтения несколькими узлами.

- **ReadWriteMany (RWX)**: том может быть смонтирован в режиме чтения-записи несколькими узлами.

Режимы доступа по StorageClass

Storage Class	Поддержка RWO	Поддержка ROX	Поддержка RWX
CephFS File Storage	Да	Нет	Да
CephRBD Block Storage	Да	Нет	Нет
TopoLVM	Да	Нет	Нет
NFS Shared Storage	Да	Нет	Да

Как показано выше, файловые системы хранения, такие как **CephFS** и **NFS**, поддерживают множественные одновременные операции записи или чтения, что делает их подходящими для сценариев совместного доступа. С другой стороны, блочные системы хранения, такие как **CephRBD** и **TopoLVM**, обеспечивают эксклюзивный доступ только одному узлу за раз.

Режимы томов в Kubernetes

Режимы томов определяют, как данные предоставляются поду:

- **Filesystem**: том монтируется в под как файловая система.
- **Block**: том представлен как необработанное блочное устройство.

Режимы томов по StorageClass

Storage Class	Тип	Поддерживаемые режимы томов
CephFS File Storage	File Storage	Filesystem

Storage Class	Тип	Поддерживаемые режимы томов
CephRBD Block Storage	Block Storage	Filesystem, Block
TopoLVM	Block Storage	Filesystem, Block
NFS Shared Storage	File Storage	Filesystem

Блочные системы хранения, такие как **CephRBD** и **TopoLVM**, предлагают как доступ через файловую систему, так и через необработанный блочный доступ, обеспечивая гибкость для различных потребностей приложений. Файловые системы хранения, такие как **CephFS** и **NFS**, напротив, поддерживают только режим файловой системы.

Особенности хранения: снимки и расширение

Kubernetes также поддерживает расширенные функции, такие как снимки томов и динамическое расширение PVC, в зависимости от используемого StorageClass.

Storage Class	Снимок тома	Расширение
CephFS File Storage	Поддерживается	Поддерживается
CephRBD Block Storage	Поддерживается	Поддерживается
TopoLVM	Поддерживается	Поддерживается
NFS Shared Storage	Не поддерживается	Не поддерживается

Снимки томов поддерживаются только для PVC, динамически выделенных с использованием StorageClass. Эта функция полезна для резервного копирования и клонирования окружений.

Заключение

При настройке хранилища в Kubernetes понимание **режимов доступа** и **режимов томов** для PVC и соответствующих **StorageClasses** критично для выбора правильного решения для вашей рабочей нагрузки. Файловые решения хранения, такие как CephFS и NFS, идеально подходят для сценариев совместного доступа, тогда как блочные системы хранения, такие как CephRBD и TopoLVM, превосходят в высокопроизводительных развертываниях на одном узле. Кроме того, поддержка таких функций, как снимки и расширение, значительно повышает гибкость хранения и стратегии управления данными.

Menu

Руководства

Создание Storage Class типа CephFS File Storage

Развертывание Volume Plugin

Создание Storage Class

Создание класса блочного хранилища CephRBD

Развертывание плагина тома

Создание класса хранилища

Создание локального Storage Class TopoLVM

Общая информация

Развертывание Volume Plugin

Создание Storage Class

Последующие действия

Создание общего класса хранения NFS

Предварительные требования

Развертывание плагина Alauda Container Platform NFS CSI

Создание общего класса хранения NFS

Развертывание компонента Volume Snapshot

Развертывание через веб-консоль

Развертывание через YAML

Создание PV

Предварительные требования

Пример PersistentVolume

Создание PV через веб-консоль

Создание PV через CLI

Связанные операции

Дополнительные ресурсы

Создание PVC

Предварительные требования

Пример PersistentVolumeClaim:

Создание Persistent Volume Claim с помощью веб-консоли

Создание Persistent Volume Claim с помощью CLI

Операции

Расширение ёмкости PersistentVolumeClaim с помощью веб-консоли

Расширение ёмкости Persistent Volume Claim с помощью CLI

Дополнительные ресурсы

Использование снимков томов

Создание Storage Class типа CephFS File Storage

CephFS file storage — это встроенная файловая система Ceph, которая предоставляет платформе метод доступа к хранилищу на основе Container Storage Interface (CSI), обеспечивая безопасный, надежный и масштабируемый общий файловый сервис, подходящий для сценариев, таких как совместное использование файлов и резервное копирование данных. Перед началом необходимо сначала создать storage class для CephFS file storage.

После привязки storage class в Persistent Volume Claim (PVC) платформа динамически создаст persistent volumes на узлах в соответствии с запросом persistent volume для бизнес-приложений.

Содержание

Развертывание Volume Plugin

После нажатия **Deploy** на странице **Distributed Storage** Создайте Storage Service или Получите доступ к Storage Service.

Создание Storage Class

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Storage Management** > **Storage Classes**.
3. Нажмите **Create Storage Class**.

   Примечание: Следующий пример приведён в форме; вы также можете создать storage class с помощью YAML.
4. Выберите **CephFS File Storage** и нажмите **Next**.
5. Настройте соответствующие параметры согласно следующим инструкциям.

Параметр	Описание
Reclaim Policy	Политика восстановления для persistent volumes. - Delete: При удалении persistent volume claim будет также удалён связанный persistent volume. - Retain: Связанный persistent volume останется, даже если persistent volume claim будет удалён.
Access Modes	Все режимы доступа, поддерживаемые текущим хранилищем. При объявлении persistent volumes можно выбрать только один из этих режимов. - ReadWriteOnce (RWO): Может быть смонтирован как для чтения и записи одним узлом. - ReadWriteMany (RWX): Может быть смонтирован для чтения и записи несколькими узлами.
Allocate Project	Укажите проекты, которым разрешено использовать этот тип хранилища. Если в данный момент нет проектов, нуждающихся в использовании этого типа хранилища, можно не выделять их сейчас и обновить позже.

Совет: Следующие параметры необходимо задать в distributed storage, они будут применены автоматически.

- Storage Cluster: Встроенный Ceph storage cluster в текущем кластере.

- Storage Pool: Логический раздел, используемый для хранения данных в storage cluster.

6. Нажмите **Create**.

Создание класса блочного хранилища CephRBD

Блочное хранилище CephRBD — это встроенное блочное хранилище Ceph для платформы, предоставляющее метод доступа к хранилищу на основе Container Storage Interface (CSI), способный обеспечивать высокие IOPS и низкую задержку, что подходит для сценариев, таких как базы данных и виртуализация. Перед использованием необходимо создать класс блочного хранилища CephRBD.

После того как Persistent Volume Claim (PVC) будет привязан к классу хранилища, платформа динамически создаст Persistent Volume на основе Persistent Volume Claim для использования бизнес-приложениями.

Содержание

Развертывание плагина тома

После нажатия **Deploy** на странице **Distributed Storage** создайте сервис хранения или подключитесь к сервису хранения.

Создание класса хранилища

1. Перейдите в раздел **Administrator**.

2. В левой навигационной панели нажмите **Storage Management** > **Storage Classes**.

3. Нажмите **Create Storage Class**.

   Примечание: Следующий пример приведён в форме, вы также можете выбрать YAML для выполнения операции.

4. Выберите **CephRBD Block Storage** и нажмите **Next**.

5. Настройте параметры согласно требованиям.

Параметр	Описание
File System	По умолчанию **EXT4** — журналируемая файловая система для Linux, способная обеспечивать хранение экстентов и обработку больших файлов. Вместимость файловой системы может достигать 1 EiB, поддерживаемый размер файла — до 16 TiB.
Reclaim Policy	Политика освобождения для persistent volume. - Delete: связанный persistent volume будет удалён вместе с persistent volume claim. - Retain: связанный persistent volume сохранится даже после удаления persistent volume claim.
Access Modes	Поддерживается только ReadWriteOnce (RWO): может быть смонтирован одним узлом в режиме чтения и записи.
Assign Project	Назначьте проекты, которые могут использовать этот тип хранилища. Если в данный момент нет проектов, нуждающихся в этом типе хранилища, можно не назначать проект и обновить позже.

   Совет: Следующие параметры необходимо задать в распределённом хранилище, они будут применены здесь напрямую.

   - Storage Cluster: встроенный Ceph storage cluster в текущем кластере.
   - Storage Pool: логический раздел, используемый для хранения данных внутри storage cluster.

6. Нажмите **Create**.

Создание локального Storage Class TopoLVM

TopoLVM — это локальное решение для хранения на базе LVM, обеспечивающее простые, удобные в обслуживании и высокопроизводительные локальные хранилища, подходящие для сценариев, таких как базы данных и middleware. Перед использованием необходимо создать Storage Class TopoLVM.

После того, как Persistent Volume Claim (PVC) будет привязан к этому Storage Class, платформа динамически создаст persistent volumes на узлах на основе PVC для использования бизнес-приложениями.

Содержание

Общая информация

Преимущества использования

- По сравнению с удалённым хранилищем (например, **NFS shared storage**): хранилище типа TopoLVM расположено локально на узле, что обеспечивает лучшую производительность по IOPS и пропускной способности, а также меньшую задержку.
- По сравнению с hostPath (например, **local-path**): хотя оба варианта являются локальным хранилищем на узле, TopoLVM позволяет гибко планировать размещение контейнерных групп на узлах с достаточными доступными ресурсами, избегая ситуации, когда контейнерные группы не могут запуститься из-за нехватки ресурсов.
- TopoLVM по умолчанию поддерживает автоматическое расширение томов. После изменения необходимой квоты хранения в Persistent Volume Claim расширение выполняется автоматически без перезапуска контейнерной группы.

Сценарии использования

- Когда требуется только временное хранилище, например, для разработки и отладки.
- При высоких требованиях к I/O хранилища, например, для индексирования в реальном времени.

Ограничения и предостережения

Рекомендуется использовать локальное хранилище только для приложений, где возможно реализовать репликацию и резервное копирование данных на уровне приложения, например, MySQL. Избегайте потери данных из-за отсутствия гарантии сохранности данных в локальном хранилище.

Узнать больше ↗

Развертывание Volume Plugin

После нажатия кнопки deploy перейдите на вновь открытую страницу configure local storage.

Создание Storage Class

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Storage Management** > **Storage Classes**.
3. Нажмите **Create Storage Class**.
4. Выберите **TopoLVM**, затем нажмите **Next**.
5. Настройте параметры Storage Class, как описано ниже.

   Примечание: Ниже приведён пример формы; вы также можете создать Storage Class с помощью YAML.

Параметр	Описание
Name	Имя Storage Class, которое должно быть уникальным в текущем кластере.
Display Name	Имя для удобства идентификации или фильтрации, например, описание Storage Class на русском языке.
Device Class	Device Class — способ классификации устройств хранения в TopoLVM, где каждый класс соответствует группе устройств с похожими характеристиками. Если нет специальных требований, используйте **Automatically Assigned**.
File System	• **XFS** — высокопроизводительная журналируемая файловая система, хорошо подходящая для параллельных I/O нагрузок, поддерживает работу с большими файлами и плавную передачу данных. • **EXT4** — журналируемая файловая система в Linux с поддержкой extent-структур, обеспечивает работу с большими файлами, максимальная ёмкость файловой системы — 1 EiB, максимальный размер файла — 16 TiB.

Параметр	Описание
Reclamation Policy	Политика освобождения persistent volumes. • Delete: связанный persistent volume будет удалён вместе с PVC. • Retain: связанный persistent volume останется даже после удаления PVC.
Access Mode	ReadWriteOnce (RWO): может быть смонтирован в режиме чтения-записи только одним узлом.
PVC Reconstruction	Поддержка реконструкции PVC на других узлах. При включении необходимо настроить **Reconstruction Wait Time**. Если узел, на котором размещён PVC, созданный с этим Storage Class, выходит из строя, PVC автоматически восстанавливается на других узлах после указанного времени ожидания для обеспечения непрерывности работы. **Примечание**: • Восстановленный PVC не содержит исходных данных. • Убедитесь, что количество узлов хранения больше количества реплик приложения, иначе это повлияет на реконструкцию PVC.
Allocated Projects	PVC этого типа можно создавать только в определённых проектах. Если в данный момент проект не назначен, его можно **обновить позже**.

6. После проверки правильности введённых данных нажмите кнопку **Create**.

Последующие действия

Когда всё будет готово, вы можете уведомить разработчиков о возможности использования функций TopoLVM. Например, создать Persistent Volume Claim и привязать его к Storage Class TopoLVM на странице **Storage** > **Persistent Volume Claims** в контейнерной платформе.

Создание общего класса хранения NFS

На основе драйвера хранения сообщества NFS CSI (Container Storage Interface) предоставляется возможность доступа к нескольким системам или аккаунтам хранения NFS.

В отличие от традиционной клиент-серверной модели доступа NFS, общее хранилище NFS использует плагин сообщества NFS CSI, который более соответствует принципам проектирования Kubernetes и позволяет клиентам обращаться к нескольким серверам.

Содержание

Предварительные требования

- Должен быть настроен NFS сервер, и необходимо получить методы доступа к нему. В настоящее время платформа поддерживает три версии протокола NFS: `v3`, `v4.0` и `v4.1`. Вы можете выполнить команду `nfsstat -s` на стороне сервера для проверки информации о версии.

Развертывание плагина Alauda Container Platform NFS CSI

Развертывание через веб-консоль

1. Войдите в систему как **Administrator**.
2. В левой навигационной панели нажмите **Storage** > **StorageClasses**.
3. Нажмите **Create StorageClass**.
4. Справа от **NFS CSI** нажмите Deploy, чтобы перейти на страницу **Plugins**.
5. Справа от плагина `Alauda Container Platform NFS CSI` нажмите ⋮ > **Install**.
6. Дождитесь, пока статус развертывания не изменится на **Deployment Successful**, после чего развертывание будет завершено.

Развертывание через YAML

См. Installing via YAML

`Alauda Container Platform NFS CSI` является **Non-config plugin**, имя модуля — `nfs`

Создание общего класса хранения NFS

1. Нажмите **Create Storage Class**.

   Примечание: Следующий контент представлен в виде формы, но вы также можете выполнить операцию с помощью YAML.

2. Выберите **NFS CSI** и нажмите **Next**.
3. Следуйте инструкциям ниже для настройки соответствующих параметров.

Параметр	Описание
Name	Имя класса хранения. Должно быть уникальным в пределах текущего кластера.
Service Address	Адрес доступа к NFS серверу. Например: `192.168.2.11`.
Path	Путь монтирования файловой системы NFS на узле сервера. Например: `/nfs/data`.
NFS Protocol Version	В настоящее время поддерживаются три версии: `v3`, `v4.0` и `v4.1`.
Reclaim Policy	Политика восстановления для постоянного тома. - Delete: При удалении запроса на постоянный том (PersistentVolumeClaim) связанный постоянный том также будет удалён. - Retain: Даже при удалении запроса на постоянный том связанный постоянный том останется.
Access Modes	Все режимы доступа, поддерживаемые текущим хранилищем. При последующем объявлении постоянных томов можно выбрать только один из этих режимов для монтирования. - ReadWriteOnce (RWO): Может быть смонтирован как для чтения и записи одним узлом. - ReadWriteMany (RWX): Может быть смонтирован как для чтения и записи несколькими узлами. - ReadOnlyMany (ROX): Может быть смонтирован как только для чтения несколькими узлами.
Allocated Projects	Пожалуйста, выделите проекты, которые могут использовать этот тип хранилища. Если в данный момент нет проектов, нуждающихся в этом типе хранилища, вы можете не выделять проекты сейчас и обновить их позже.
subDir	Каждый PersistentVolumeClaim (PVC), созданный с использованием общего класса хранения NFS, соответствует

Параметр	Описание
	подкаталогу внутри общего ресурса NFS. По умолчанию подкаталоги именуются по шаблону `${pv.metadata.name}` (то есть именем PersistentVolume). Если имя, сгенерированное по умолчанию, не соответствует вашим требованиям, вы можете настроить правила именования подкаталогов.

NOTE

Поле `subDir` поддерживает только следующие три переменные, которые автоматически разрешаются драйвером NFS CSI:

- `${pvc.metadata.namespace}` : пространство имён PVC.
- `${pvc.metadata.name}` : имя PVC.
- `${pv.metadata.name}` : имя PV.

Правило именования `subDir` **ДОЛЖНО** обеспечивать уникальность имён подкаталогов. В противном случае несколько PVC могут использовать один и тот же подкаталог, что приведёт к конфликтам данных.

Рекомендуемые конфигурации:

- `${pvc.metadata.namespace}_${pvc.metadata.name}_${pv.metadata.name}`
- `<cluster-identifier>_${pvc.metadata.namespace}_${pvc.metadata.name}_${pv.metadata.name}`

Предназначено для нескольких Kubernetes кластеров, использующих один и тот же NFS сервер. Такая конфигурация обеспечивает чёткое различие кластеров за счёт включения идентификатора кластера (например, имени кластера) в правила именования подкаталогов.

Не рекомендуемые конфигурации:

- `${pvc.metadata.namespace}-${pvc.metadata.name}-${pv.metadata.name}` Не используйте дефис (-) в качестве разделителя, так как это может привести к неоднозначным именам подкаталогов. Например: если два PVC называются `ns-1/test` и `ns/1-test`, оба могут сгенерировать одинаковый подкаталог `ns-1-test`.

- `${pvc.metadata.namespace}/${pvc.metadata.name}/${pv.metadata.name}` НЕ настраивайте subDir для создания вложенных директорий. Драйвер NFS CSI удаляет только последний уровень директории `${pv.metadata.name}` при удалении PVC, оставляя родительские директории сиротами на NFS сервере.

4. После подтверждения правильности конфигурации нажмите **Create**.

Развертывание компонента Volume Snapshot

Volume snapshot — это снимок persistent volume, представляющий собой копию persistent volume на определённый момент времени. Если в кластере используются persistent volumes с поддержкой функции snapshot, можно развернуть компонент volume snapshot для включения этой возможности.

В настоящее время платформа поддерживает создание volume snapshots только для PVC, которые **динамически создаются** с использованием storage classes. На основе этих снимков можно создавать новые привязки PVC.

Совет: Режимы доступа, поддерживаемые при создании PVC из снимков, отличаются от тех, что поддерживаются при создании PVC с помощью storage classes, и выделены **жирным** в таблице ниже.

Storage Class, используемый для создания Volume Snapshots	Single Node Read-Write (RWO)	Multi-Node Read-Only (ROX)	Multi-Node Read-Write (RWX)
TopoLVM	Поддерживается	Не поддерживается	Не поддерживается
CephRBD Block Storage	Поддерживается	Не поддерживается	Не поддерживается
CephFS File Storage	Поддерживается	**Поддерживается**	Поддерживается

Содержание

Развертывание через веб-консоль

1. Перейдите в раздел **Administrator**.
2. Нажмите **Marketplace** > **Cluster Plugins**, чтобы открыть страницу списка **Cluster Plugins**.
3. Найдите плагин кластера `Alauda Container Platform Snapshot Management`, нажмите **Install** и дождитесь успешного завершения развертывания.

Развертывание через YAML

См. раздел Installing via YAML

`Alauda Container Platform Snapshot Management` — это **Non-config plugin**, имя модуля — `snapshot`

Создание PV

Ручное создание статического persistent volume типа **HostPath** или **NFS Shared Storage**.

- **HostPath**: монтирует файловый каталог с хоста, на котором находится контейнер, в указанный путь внутри контейнера (соответствует HostPath в Kubernetes), позволяя контейнеру использовать файловую систему хоста для постоянного хранения. Если хост становится недоступен, HostPath может стать недоступен.
- **NFS Shared Storage**: NFS Shared Storage использует сообщественный плагин хранения NFS CSI (Container Storage Interface), который более соответствует принципам проектирования Kubernetes, обеспечивая возможность клиентского доступа для нескольких сервисов. Перед использованием убедитесь, что в текущем кластере развернут **NFS storage plugin**.

Содержание

Предварительные требования

- Подтвердите размер создаваемого persistent volume и убедитесь, что бэкенд-хранилище в данный момент имеет возможность предоставить соответствующий объем.
- Получите адрес доступа к бэкенд-хранилищу, путь к монтируемой директории, учетные данные доступа (если требуются) и другую необходимую информацию.

Пример PersistentVolume

```yaml
# example-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi (1)
  accessModes:
    - ReadWriteOnce (2)
  persistentVolumeReclaimPolicy: Retain (3)
  storageClassName: manual (4)
  hostPath: (5)
    path: "/mnt/data"
```

(1) Объем хранилища.

(2) Способ монтирования тома.

(3) Что происходит после удаления PVC (Retain, Delete, Recycle).

(4) Имя StorageClass (для динамического связывания).

(5) Тип бэкенд-хранилища.

Создание PV через веб-консоль

1. Перейдите в раздел **Administrator**.

2. В левой навигационной панели выберите **Storage Management** > **Persistent Volumes (PV)**.
3. Нажмите **Create Persistent Volume**.
4. Ознакомьтесь с инструкциями ниже и настройте параметры перед нажатием **Create**.

Информация о хранилище

Тип	Параметр	Описание
HostPath	Path	Путь к каталогу файлов на узле, который поддерживает том хранилища. Например: `/etc/kubernetes`.
NFS Shared Storage	Server Address	Адрес доступа к NFS-серверу.
	Path	Путь монтирования файловой системы NFS на серверном узле, например `/nfs/data`.
	NFS Protocol Version	Поддерживаемые версии протокола NFS на платформе: `v3`, `v4.0` и `v4.1`. Для просмотра информации о версиях на сервере выполните команду `nfsstat -s`.

Создание PV через CLI

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

Режимы доступа

Режимы доступа persistent volume зависят от соответствующих параметров, установленных бэкенд-хранилищем.

Режим доступа	Значение
ReadWriteOnce (RWO)	Может быть смонтирован с правами чтения и записи одним узлом.
ReadWriteMany (RWX)	Может быть смонтирован с правами чтения и записи несколькими узлами.
ReadOnlyMany (ROX)	Может быть смонтирован только для чтения несколькими узлами.

Политики возврата

Политика возврата	Значение
Delete	При удалении persistent volume claim одновременно удаляется связанный persistent volume, а также ресурс тома в бэкенд-хранилище. **Примечание**: политика возврата для PV типа NFS Shared Storage не поддерживает **Delete**.
Retain	Даже при удалении persistent volume claim связанный persistent volume и данные хранилища сохраняются. Требуется ручное управление данными и последующее удаление persistent volume.

Связанные операции

Вы можете нажать ⋮ справа на странице списка или выбрать **Operations** в правом верхнем углу страницы с деталями для обновления или удаления persistent volume по необходимости.

Удаление persistent volume применимо в следующих двух сценариях:

- Удаление несвязанного persistent volume: том не использовался для записи и больше не требуется для записи, что освобождает соответствующее пространство хранилища

при удалении.

- Удаление persistent volume с политикой **Retain**: persistent volume claim был удалён, но из-за политики retain том не был удалён одновременно. Если данные persistent volume были сохранены в другом хранилище или больше не нужны, удаление тома также освободит соответствующее пространство.

Дополнительные ресурсы

- Creating PVCs

Создание PVC

Создайте PersistentVolumeClaim (PVC) и при необходимости задайте параметры для запрашиваемого PersistentVolume (PV).

Вы можете создать PersistentVolumeClaim либо через визуальную форму UI, либо с помощью пользовательского YAML-файла оркестрации.

Содержание

Предварительные требования

Убедитесь, что в namespace достаточно оставшейся квоты **хранилища** для удовлетворения требуемого размера хранилища при выполнении операции создания.

Пример PersistentVolumeClaim:

```yaml
# example-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
  namespace: k-1
  annotations: {}
  labels: {}
spec:
  storageClassName: cephfs
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 4Gi
```

Создание Persistent Volume Claim с помощью веб-консоли

1. Перейдите в **Container Platform**.
2. В левой боковой панели выберите **Storage > PersistentVolumeClaims (PVC)**.
3. Нажмите **Create PVC**.
4. Настройте параметры по необходимости.

   Примечание: Следующий контент приведён в качестве примера с использованием формы; вы также можете переключиться в режим YAML для выполнения операции.

Параметр	Описание
Name	Имя PersistentVolumeClaim, которое должно быть уникальным в пределах текущего namespace.

Параметр	Описание
Creation Method	- Dynamic Creation: Динамически создаёт PersistentVolume на основе storage class и связывает его. - Static Binding: Выполняет сопоставление и связывание на основе настроенных параметров и существующих PersistentVolumes.
Storage Class	После выбора метода динамического создания платформа динамически создаст PersistentVolume в соответствии с описанием в указанном storage class.
Access Mode	- ReadWriteOnce (RWO): Может быть смонтирован одним узлом в режиме чтения и записи. - ReadWriteMany (RWX): Может быть смонтирован несколькими узлами в режиме чтения и записи. - ReadOnlyMany (ROX): Может быть смонтирован несколькими узлами в режиме только для чтения. **Совет**: Рекомендуется учитывать количество экземпляров workload, которые планируется связать с текущим PersistentVolumeClaim, а также тип контроллера развертывания. Например, при создании мультиэкземплярного развертывания (Deployment), поскольку все экземпляры используют один PersistentVolumeClaim, не рекомендуется выбирать режим доступа RWO, который может быть подключён только к одному узлу.
Capacity	Размер запрашиваемого PersistentVolume.
Volume Mode	- Filesystem: Связывает PersistentVolume как файловую директорию, монтируемую в Pod. Этот режим доступен для любого типа workload. - Block Device: Связывает PersistentVolume как необработанное блочное устройство, монтируемое в Pod. Этот режим доступен только для виртуальных машин.
More	- Labels - Annotations

Параметр	Описание
	- Selector: После выбора метода статического связывания можно использовать селектор для выбора PersistentVolumes с определёнными метками. Метки PersistentVolume могут использоваться для обозначения специальных атрибутов хранилища, таких как тип диска или географическое расположение.

5. Нажмите **Create**. Дождитесь, пока статус PersistentVolumeClaim не изменится на `Bound`, что означает успешное сопоставление PersistentVolume.

Создание Persistent Volume Claim с помощью CLI

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

Операции

- **Связывание PersistentVolumeClaim**: При создании приложений или workload, требующих постоянного хранения данных, свяжите PersistentVolumeClaim для запроса соответствующего PersistentVolume.
- **Создание PersistentVolumeClaim с использованием Volume Snapshots**: Это помогает создавать резервные копии данных приложений и восстанавливать их по необходимости, обеспечивая надёжность данных бизнес-приложений. Пожалуйста, обратитесь к Using Volume Snapshots.
- **Удаление PersistentVolumeClaim**: Вы можете нажать кнопку **Actions** в правом верхнем углу страницы с деталями для удаления PersistentVolumeClaim при необходимости. Перед удалением убедитесь, что PersistentVolumeClaim не связан с какими-либо приложениями или workload и не содержит volume snapshots. После удаления PersistentVolumeClaim платформа обработает PersistentVolume в соответствии с политикой рекламации, что может привести к очистке данных в

PersistentVolume и освобождению ресурсов хранилища. Пожалуйста, действуйте осторожно, учитывая безопасность данных.

Расширение ёмкости PersistentVolumeClaim с помощью веб-консоли

1. В левой навигационной панели выберите Storage > Persistent Volume Claims (PVC).
2. Найдите нужный persistent volume claim и нажмите ⋮ > Expand.
3. Укажите новый размер.
4. Нажмите Expand. Процесс расширения может занять некоторое время, пожалуйста, будьте терпеливы.

Расширение ёмкости Persistent Volume Claim с помощью CLI

```
kubectl patch pvc example-pvc -n k-1 --type='merge' -p '{
  "spec": {
    "resources": {
      "requests": {
        "storage": "6Gi"
      }
    }
  }
}'
```

INFO

Если расширение PVC в Kubernetes не удалось, администраторы могут вручную восстановить состояние Persistent Volume Claim (PVC) и отменить запрос на расширение. См. Recover From PVC Expansion Failure

Дополнительные ресурсы

- How to Annotate Third-Party Storage Capabilities

Использование снимков томов

Снимок тома — это копия persistent volume claim (PVC) на определённый момент времени, которая может использоваться для настройки новых persistent volume claim (предварительное заполнение данными из снимка) или для отката существующих persistent volume claim к предыдущему состоянию, достигая эффекта резервного копирования данных приложения и их восстановления по необходимости, тем самым обеспечивая надёжность данных приложения.

Содержание

Предварительные требования

- Администратор развернул компонент снимков томов **Snapshot Controller** для текущего кластера и включил функции, связанные со снимками, в кластере хранения.

- Persistent volume claim должен быть создан динамически, а его статус должен быть **Bound**.
- Storage class, связанный с persistent volume claim, должен поддерживать функциональность снимков, например, **CephRBD Built-in Storage**, **CephFS Built-in Storage** или **TopoLVM**.

Пример ресурса VolumeSnapshot (CR)

Этот пример создаёт снимок PVC с именем example-pvc с использованием CSI snapshot class.

```
# example-snapshot.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-pvc-20250527-111124
  namespace: k-1
  labels:
    snapshot.cpaas.io/sourcepvc: example-pvc
  annotations:
    cpaas.io/description: demo
spec:
  volumeSnapshotClassName: csi-cephfs-snapshotclass
  source:
    persistentVolumeClaimName: example-pvc
```

Создание снимков томов через веб-консоль

Создание снимка тома на основе указанного persistent volume claim (PVC)

Способ первый

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Storage > Persistent Volume Claims (PVC)**.
3. Нажмите ⋮ рядом с соответствующим persistent volume claim в списке и выберите **Create Volume Snapshot**.
4. Заполните описание снимка. Это описание поможет зафиксировать текущее состояние persistent volume, например *Перед обновлением приложения*.
5. Нажмите **Create**. Время создания снимка зависит от состояния сети и объёма данных; пожалуйста, подождите.

   Когда статус снимка изменится на `Available`, это означает успешное создание.

Способ второй

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Storage > Persistent Volume Claims (PVC)**.
3. Нажмите на имя persistent volume claim в списке.
4. Перейдите на вкладку **Volume Snapshots**.
5. Нажмите **Create Volume Snapshot** и настройте необходимые параметры.
6. Нажмите **Create**. Время создания снимка зависит от состояния сети и объёма данных; пожалуйста, подождите.

   Когда статус снимка изменится на `Available`, это означает успешное создание.

Создание снимков томов в произвольном порядке

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Storage > Volume Snapshots**.
3. Нажмите **Create Volume Snapshot** и настройте необходимые параметры.
4. Нажмите **Create**. Время создания снимка зависит от состояния сети и объёма данных; пожалуйста, подождите.

   Когда статус снимка изменится на `Available`, это означает успешное создание.

Создание снимков томов через CLI

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

Создание persistent volume claims из снимков томов

В настоящее время платформа поддерживает создание снимков томов только для PVC, созданных из storage class с **Dynamic Provisioning**. Вы можете создавать новые PVC на основе этих снимков и связывать их.

Примечание: Режимы доступа, поддерживаемые при создании PVC из снимка, отличаются от поддерживаемых при создании PVC из storage class, что выделено **жирным** в таблице.

Storage Class, используемый для создания снимков томов	Single Node Read-Write (RWO)	Multi-Node Read-Only (ROX)	Multi-Node Read-Write (RWX)
TopoLVM	Поддерживается	Не поддерживается	Не поддерживается
CephRBD Block Storage	Поддерживается	Не поддерживается	Не поддерживается
CephFS File Storage	Поддерживается	**Поддерживается**	Поддерживается

Способ первый

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Storage > Persistent Volume Claims (PVC)**.
3. Нажмите на имя persistent volume claim в списке.

4. Перейдите на вкладку **Volume Snapshots**.
5. Нажмите ⋮ рядом с соответствующим снимком тома в списке и выберите **Create Persistent Volume Claim**, настройте необходимые параметры.
6. Нажмите **Create**.

Способ второй

1. Войдите в **Container Platform**.
2. В левой навигационной панели выберите **Storage > Volume Snapshots**.
3. Нажмите ⋮ рядом с соответствующим снимком тома в списке и выберите **Create Persistent Volume Claim**, настройте необходимые параметры.
4. Нажмите **Create**.

Дополнительный ресурс

- Создание PVC

Menu

Как сделать

Generic ephemeral volumes

Пример ephemeral volumes

Основные характеристики

Когда использовать Generic Ephemeral Volumes

Чем они отличаются от emptyDir?

Использование emptyDir

Пример emptyDir

Необязательная настройка Medium

Основные характеристики

Распространённые сценарии использования

Настройка постоянного хранилища с использованием NFS

Предварительные требования

Процедура

Принудительное ограничение квот диска через разделённые экспорты

Безопасность NFS томов

Освобождение ресурсов

Руководство по аннотированию возможностей стороннего хранилища

Generic ephemeral volumes

Generic Ephemeral Volumes в Kubernetes — это функция, которая позволяет создавать эфемерные (временные) тома на уровне пода с использованием существующих StorageClasses и CSI-драйверов, без необходимости предварительного определения PersistentVolumeClaims (PVC).

Они сочетают гибкость динамического выделения с простотой объявления томов на уровне пода.

- Это временные тома, которые автоматически:
  - создаются при запуске Pod
  - удаляются при завершении Pod
- Используют те же базовые механизмы, что и PersistentVolumeClaim
- Требуют CSI (Container Storage Interface) драйвер с поддержкой динамического выделения

Содержание

Пример ephemeral volumes

Этот пример автоматически создаёт временный PVC для Pod с использованием указанного `StorageClass`.

```
apiVersion: v1
kind: Pod
metadata:
  name: ephemeral-demo
spec:
  containers:
    - name: app
      image: busybox
      command: ["sh", "-c", "echo hello > /data/hello.txt && sleep 3600"]
      volumeMounts:
        - mountPath: /data
          name: ephemeral-volume
  volumes:
    - name: ephemeral-volume
      ephemeral: (1)
        volumeClaimTemplate:
          metadata:
            labels:
              type: temporary
          spec:
            accessModes: [ "ReadWriteOnce" ]
            resources:
              requests:
                storage: 1Gi
            storageClassName: standard
```

(1) `Pod` создаст `PVC`, используя этот шаблон.

Основные характеристики

Характеристика	Описание
Эфемерный	Том удаляется при удалении Pod

Характеристика	Описание
Динамическое выделение	Поддерживается любым CSI-драйвером с динамическим выделением
Без отдельного PVC	VolumeClaim встроен непосредственно в спецификацию Pod
Работает через CSI	Совместим с любым CSI-драйвером (EBS, RBD, Longhorn и др.)

Когда использовать Generic Ephemeral Volumes

- Когда требуется временное хранилище с такими возможностями, как:
  - Изменяемый размер томов
  - Снимки (snapshots)
  - Шифрование
  - Хранилище, не привязанное к локальному узлу (например, облачное блочное хранилище)
- Идеально подходит для:
  - Кэширования промежуточных данных
  - Временных рабочих директорий
  - Конвейеров, AI/ML рабочих процессов

Чем они отличаются от emptyDir?

Характеристика	`emptyDir`	Generic Ephemeral Volume
Бэкенд хранилища	Локальный диск или память узла	Любой бэкенд с поддержкой CSI
Возможности хранения	Базовые	Поддержка снимков, шифрования и др.
Сценарий использования	Простое временное хранилище	Расширенные требования к эфемерному хранилищу
Возможность пересоздания	Нет (привязан к узлу)	Да (если CSI-том можно подключить)

Использование emptyDir

В Kubernetes emptyDir — это простой временный тип тома, который предоставляет временное хранилище для пода на время его жизни. Он создаётся, когда под назначается на узел, и удаляется, когда под удаляется с этого узла.

Содержание

Пример emptyDir

Этот Pod создаёт временный том, смонтированный в /data, который используется контейнером.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: emptydir-demo
spec:
  containers:
    - name: app
      image: busybox
      command: ["sh", "-c", "echo hello > /data/hello.txt && sleep 3600"]
      volumeMounts:
        - mountPath: /data
          name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

Необязательная настройка Medium

Вы можете выбрать, где будут храниться данные:

```yaml
emptyDir:
  medium: "Memory"
```

Medium	Описание
(default)	Использует диск узла, SSD или сетевое хранилище, в зависимости от среды
`Memory`	Использует RAM (`tmpfs`) для более быстрого доступа (но данные нестабильны)

Основные характеристики

Особенность	Описание
Начинается пустым	Нет данных при создании
Совместное использование	Один и тот же том может использоваться несколькими контейнерами в поде
Удаляется с подом	Том уничтожается при удалении пода
Локальный для узла	Том хранится на локальном диске или в памяти узла
Быстрый	Идеально подходит для производительного временного хранилища

Распространённые сценарии использования

- Кэширование промежуточных артефактов сборки
- Буферизация логов
- Временные рабочие директории
- Совместное использование данных между контейнерами в одном поде (например, сайдкары)

Настройка постоянного хранилища с использованием NFS

Кластеры Alauda Container Platform поддерживают постоянное хранилище с использованием NFS. Persistent Volumes (PV) и Persistent Volume Claims (PVC) обеспечивают уровень абстракции для предоставления и использования томов хранения внутри проекта. Хотя детали конфигурации NFS можно встроить непосредственно в определение Pod, такой подход не создаёт том как отдельный, изолированный ресурс кластера, что увеличивает риск конфликтов.

Содержание

Предварительные требования

- Хранилище должно существовать в базовой инфраструктуре до того, как его можно будет примонтировать как том в Alauda Container Platform.
- Для предоставления NFS томов требуется только список NFS серверов и путей экспорта.

Процедура

1 Создайте определение объекта для PV

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-example 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 10.0.0.3 6
  persistentVolumeReclaimPolicy: Retain 7
EOF
```

1 Имя тома.

2 Объём хранилища.

3 Хотя это кажется связанным с контролем доступа к тому, на самом деле это используется аналогично меткам и служит для сопоставления PVC с PV. В настоящее время правила доступа на основе accessModes не применяются.

4 Тип используемого тома, в данном случае плагин nfs.

5 Адрес NFS сервера.

6 Путь экспорта NFS.

7 Что происходит после удаления PVC (Retain, Delete, Recycle).

2 Проверьте, что PV был создан

Команда

```
kubectl get pv
```

Пример вывода

```
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
pv-nfs-example   1Gi        RWO            Retain           Available
10s
```

3 Создайте PVC, ссылающийся на PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
  - ReadWriteOnce (1)
  resources:
    requests:
      storage: 1Gi (2)
  volumeName: pv-nfs-example (3)
  storageClassName: ""
```

(1) Режимы доступа не обеспечивают безопасность, а служат метками для сопоставления PV с PVC.

(2) Этот запрос ищет PV с объёмом 1Gi или больше.

(3) Имя PV, который будет использоваться.

4 Проверьте, что persistent volume claim был создан

Команда

```
kubectl get pvc
```

Пример вывода

```
NAME           STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS
AGE
nfs-claim1     Bound    pv-nfs-example   1Gi        RWO
10s
```

Принудительное ограничение квот диска через разделённые экспорты

Для применения квот диска и ограничений по размеру можно использовать дисковые разделы. Назначьте каждый раздел как отдельную точку экспорта, при этом каждый экспорт соответствует отдельному PersistentVolume (PV).

Хотя Alauda Container Platform требует уникальных имён PV, ответственность за обеспечение уникальности сервера и пути NFS тома для каждого экспорта лежит на администраторе.

Такой подход с разделами позволяет точно управлять ёмкостью. Разработчики запрашивают постоянное хранилище с указанием требуемого объёма (например, 10Gi), и ACP сопоставляет запрос с PV, поддерживаемым разделом/экспортом, предлагающим не менее указанного объёма. Обратите внимание: ограничение квоты применяется к доступному объёму хранилища внутри назначенного раздела/экспорта.

Безопасность NFS томов

В этом разделе описываются механизмы безопасности NFS томов с акцентом на сопоставление прав доступа. Предполагается, что читатели обладают базовыми знаниями о POSIX-правах, UID процессов и дополнительных группах.

Разработчики запрашивают NFS хранилище через:

- Ссылку PersistentVolumeClaim (PVC) по имени, или

- Прямую конфигурацию плагина NFS тома в разделе volumes спецификации Pod.

На NFS сервере файл /etc/exports определяет правила экспорта для доступных директорий. Каждая экспортируемая директория сохраняет свои исходные POSIX идентификаторы владельца/группы.

Ключевое поведение плагина NFS в Alauda Container Platform:

1. Монтирует тома в контейнеры, сохраняя точные POSIX права и владельца исходной директории
2. Запускает контейнеры без принудительного сопоставления UID процессов с владельцем монтирования — это преднамеренная мера безопасности

Например, рассмотрим NFS директорию с такими атрибутами на сервере:

Команда

```
ls -l /share/nfs -d
```

Пример вывода

```
drwxrws---. nfsnobody 5555   /share/nfs
```

Команда

```
id nfsnobody
```

Пример вывода

```
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

Тогда контейнер должен либо запускаться с UID 65534, владельцем nfsnobody, либо иметь 5555 в дополнительных группах (supplemental groups) для доступа к директории.

NOTE

Примечание ID владельца 65534 приведён в качестве примера. Несмотря на то, что root_squash NFS отображает root (uid 0) в nfsnobody (uid 65534), NFS экспорты могут иметь

произвольные ID владельцев. Владелец 65534 не обязателен для NFS экспортов.

Идентификаторы групп

Рекомендуемое управление доступом к NFS (когда права экспорта фиксированы)
Если изменить права на NFS экспорте невозможно, рекомендуемый способ управления доступом — через дополнительные группы.

Дополнительные группы в Alauda Container Platform — распространённый механизм контроля доступа к общему файловому хранилищу, такому как NFS.

В отличие от блочного хранилища: доступ к блочным томам (например, iSCSI) обычно управляется установкой значения fsGroup в securityContext пода. Этот подход использует изменение групповой принадлежности файловой системы при монтировании.

NOTE

Для доступа к постоянному хранилищу обычно предпочтительнее использовать дополнительные групповые ID, а не пользовательские ID.

Поскольку групповой ID целевой NFS директории в примере равен 5555, под может определить этот групповой ID через supplementalGroups в securityContext пода. Например:

```
spec:
  containers:
    - name:
    ...
  securityContext: (1)
    supplementalGroups: [5555] (2)
```

(1) securityContext должен быть определён на уровне пода, а не внутри конкретного контейнера.

(2) Массив GID, определённых для пода. В данном случае массив содержит один элемент. Дополнительные GID разделяются запятыми.

Идентификаторы пользователей

UID могут быть определены в образе контейнера или в определении Pod.

> **NOTE**
>
> Для доступа к постоянному хранилищу обычно предпочтительнее использовать дополнительные групповые ID, а не пользовательские ID.

В приведённом выше примере целевой NFS директории контейнеру нужно установить UID 65534, игнорируя пока групповые ID, поэтому в определение Pod можно добавить:

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

1 В Pod есть определение securityContext для каждого контейнера и общий securityContext пода, который применяется ко всем контейнерам в поде.

2 65534 — это пользователь nfsnobody.

Настройки экспорта

Чтобы разрешить произвольным пользователям контейнера читать и записывать том, каждый экспортируемый том на NFS сервере должен соответствовать следующим условиям:

- Каждый экспорт должен быть экспортирован в следующем формате:

```
# замените 10.0.0.0/24 на доверенные CIDR/хосты
/<example_fs> 10.0.0.0/24(rw,sync,root_squash,no_subtree_check)
```

- Межсетевой экран должен быть настроен на разрешение трафика к точке монтирования.

- Для NFSv4 настройте порт по умолчанию 2049 (nfs).

```
iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

- Для NFSv3 необходимо настроить три порта: 2049 (nfs), 20048 (mountd) и 111 (portmapper).

```
iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- Экспорт NFS и директория должны быть настроены так, чтобы они были доступны целевым подам. Либо установите владельцем экспорта основной UID контейнера, либо предоставьте доступ группе пода через supplementalGroups, как показано выше для групповых ID.

Освобождение ресурсов

NFS реализует интерфейс Recyclable плагина Alauda Container Platform. Автоматические процессы обрабатывают задачи освобождения ресурсов на основе политик, установленных для каждого persistent volume.

По умолчанию PV настроены на Retain.

После удаления претензии на PVC и освобождения PV объект PV не должен повторно использоваться. Вместо этого следует создать новый PV с теми же основными параметрами тома, что и у исходного.

Например, администратор создаёт PV с именем nfs1:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

Пользователь создаёт PVC1, который связывается с nfs1. Затем пользователь удаляет PVC1, освобождая претензию на nfs1. В результате nfs1 получает статус Released. Если администратор хочет сделать тот же NFS шар доступным, он должен создать новый PV с теми же деталями NFS сервера, но с другим именем PV:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

Удаление исходного PV и повторное создание с тем же именем не рекомендуется. Попытка вручную изменить статус PV с Released на Available вызывает ошибки и потенциальную потерю данных.

Руководство по аннотированию возможностей стороннего хранилища

> **Обзор функции:** Добавляя ConfigMap **StorageDescription** в пространство имён `kube-public`, платформа автоматически определяет поддержку снимков (snapshot) для каждого стороннего StorageClass, а также поддерживаемые режимы томов и режимы доступа (включая режимы доступа, специфичные для блочных томов). Экран создания PVC затем отображает только допустимые варианты, помогая легко выбрать и использовать нужные возможности хранилища.

Содержание

---

1. Начало работы

1.1 Создание или обновление ConfigMap

> **Важно:** Выполняйте следующую операцию **в пространстве имён** `kube-public`, иначе платформа не распознает возможности хранилища.

Отредактируйте или создайте ConfigMap с именем, начинающимся с `sd-`, например `sd-capabilities-enhanced`:

```
kubectl -n kube-public edit configmap sd-capabilities-enhanced
```

Обязательная метка

```
metadata:
  labels:
    features.alauda.io/type: StorageDescription
```

1.2 Заполнение поля `data`

Каждый `key` соответствует `provisioner` StorageClass; значение — это YAML-строка, описывающая его возможности. Основные поля:

Поле	Тип	Описание
`snapshot`	`Boolean`	Указывает, поддерживаются ли снимки томов
`volumeMode`	`List[String]`	Поддерживаемые режимы томов; минимум один из `Filesystem`, `Block`
`accessModes`	`List[String]`	Режимы доступа, доступные при `volumeMode` равном `Filesystem`
`blockAccessModes`	`List[String]`	Режимы доступа, специфичные для блочных томов (опционально)

> Если `blockAccessModes` отсутствует, платформа будет использовать `accessModes` для блочных томов.

1.3 Применение конфигурации

```
kubectl apply -f sd-capabilities-enhanced.yaml
```

После применения UI автоматически подстраивает доступные опции, например:

- При выборе режима тома **Block** выпадающий список режимов доступа заполняется значениями из `blockAccessModes`.
- Если `snapshot: true`, операции, связанные со снимками, становятся доступны на странице PVC.

---

2. Пример ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: sd-capabilities-enhanced
  namespace: kube-public
  labels:
    features.alauda.io/type: StorageDescription
data:
  storage.advanced-block-fs.com: |-
    snapshot: true
    volumeMode:
      - Filesystem
      - Block
    accessModes:
      - ReadWriteOnce
      - ReadOnlyMany
    blockAccessModes:
      - ReadWriteOnce
  storage.filesystem-basic.com: |-
    snapshot: false
    volumeMode:
      - Filesystem
    accessModes:
      - ReadWriteOnce
      - ReadWriteMany
```

3. Обновление существующих описаний возможностей

1. Найдите ключ `provisioner`, который хотите изменить.
2. Отредактируйте значения полей, чтобы они отражали актуальные возможности.
3. Повторно примените ConfigMap с помощью `kubectl apply -f ...`. Платформа опрашивает обновления и автоматически обновляет UI; вы также можете обновить страницу браузера для немедленного отображения изменений.

---

4. Совместимость с устаревшим форматом

- Если `blockAccessModes` отсутствует, блочные тома наследуют `accessModes`.
- Удалять старые ConfigMap не нужно; просто добавьте новые поля для плавного обновления.

---

5. Часто задаваемые вопросы

Симптом	Возможная причина	Решение
Список режимов доступа пуст для блочных томов	`blockAccessModes` пуст и `accessModes` также пусты	Укажите хотя бы одно из двух
UI по-прежнему показывает устаревшие возможности	ConfigMap не сохранён или кэш браузера	Проверьте с помощью `kubectl get cm`, обновите страницу

Menu

Устранение неполадок

Восстановление после ошибки расширения PVC

Когда расширение PVC в Kubernetes завершается неудачей, администраторы могут вручную восстановить состояние Persistent Volume Claim (PVC) и отменить запрос на расширение.

Содержание

Процедура

1. Измените политику повторного использования (reclaim policy) Persistent Volume (PV), связанного с PVC, на `Retain`. Для этого отредактируйте соответствующий PV и установите поле `persistentVolumeReclaimPolicy` в значение `Retain`.
2. Удалите исходный PVC.
3. Вручную отредактируйте PV, удалив запись `claimRef` из его спецификации. Это обеспечит возможность привязки нового PVC к этому PV, изменив статус PV на `Available`.
4. Воссоздайте новый PVC с меньшим размером или размером, поддерживаемым базовым поставщиком хранилища.
5. Явно укажите поле `volumeName` в новом PVC, чтобы оно совпадало с именем исходного PV. Это гарантирует точную привязку нового PVC к указанному PV.

6. Наконец, восстановите исходную политику повторного использования PV.

Дополнительные рекомендации

- Убедитесь, что используемый `StorageClass` поддерживает расширение томов, установив `allowVolumeExpansion` в `true`.
- Выполняйте эти действия внимательно, чтобы избежать риска потери данных.