Расширение

Обзор

Платформа предоставляет комплексную систему расширений, которая позволяет пользователям расширять функциональность своих Kubernetes кластеров. Эта система разработана с учетом гибкости и удобства использования, что позволяет легко добавлять новые функции и возможности в кластеры.

Система состоит из двух основных типов расширений:

- Operators: Operators построены на базе фреймворка Operator Lifecycle Manager (OLM) v0 и обеспечивают специализированные операционные возможности для платформы. Эти расширения позволяют автоматизировать управление сложными приложениями и сервисами внутри вашего кластера.
- Cluster Plugins: Платформа включает собственную систему кластерных плагинов, специально разработанную для плагинов типа Chart. Эта система обеспечивает улучшенный опыт установки и управления по сравнению со стандартными методами, предлагая удобный интерфейс для работы с расширениями на основе Chart.

Благодаря поддержке множества Operators и кластерных плагинов пользователи могут значительно расширить возможности платформы для удовлетворения конкретных операционных требований и сценариев использования.

Operator

Содержание

Обзор

На основе фреймворка **OLM (Operator Lifecycle Manager)**, **OperatorHub** предоставляет единый интерфейс для управления установкой, обновлением и жизненным циклом Операторов.
Администраторы могут использовать OperatorHub для установки и управления Операторами, обеспечивая полную автоматизацию жизненного цикла приложений Kubernetes, включая создание, обновления и удаление.

OLM в основном состоит из следующих компонентов и CRD:

- **OLM (olm-operator)**: Управляет полным жизненным циклом Операторов, включая установку, обновления и обнаружение конфликтов версий.
- **Catalog Operator**: Управляет каталогами Операторов и генерирует соответствующие InstallPlan.
- **CatalogSource**: CRD с областью видимости namespace, который управляет источником каталога Операторов и предоставляет метаданные Оператора (например, информацию о версиях, управляемые CRD). Платформа предоставляет 3 стандартных CatalogSource: **system**, **platform** и **custom**. Операторы из **system** не отображаются в OperatorHub.
- **ClusterServiceVersion (CSV)**: CRD с областью видимости namespace, описывающий конкретную версию Оператора, включая необходимые ресурсы, CRD и разрешения.
- **Subscription**: CRD с областью видимости namespace, описывающий подписанный Оператор, его источник, канал получения и стратегию обновления.
- **InstallPlan**: CRD с областью видимости namespace, описывающий фактические операции установки (например, создание Deployments, CRD, RBAC). Оператор будет установлен или обновлен только после одобрения InstallPlan.

Источники Операторов

Для уточнения стратегии жизненного цикла различных Операторов в OperatorHub платформа предоставляет 5 типов источников:

1. **Alauda**
   Предоставляется и поддерживается Alauda, включая полное управление жизненным циклом, обновления безопасности, техническую поддержку и SLA.
2. Curated
   Отобранные из сообщества с открытым исходным кодом, соответствующие версиям сообщества, без модификации кода или перекомпиляции. Alauda предоставляет рекомендации и обновления безопасности, но не гарантирует SLA или управление жизненным циклом.
3. Community
   Предоставляются сообществом с открытым исходным кодом, обновляются периодически для обеспечения возможности установки, но функциональная полнота не гарантируется; SLA и поддержка Alauda отсутствуют.
4. Marketplace
   Предоставляются и поддерживаются сторонними поставщиками, сертифицированными Alauda. Alauda обеспечивает поддержку интеграции с платформой, а поставщик отвечает за основное сопровождение.
5. Custom
   Разрабатываются и загружаются пользователем для удовлетворения индивидуальных требований.

Подготовка к установке

Перед установкой Оператора необходимо ознакомиться со следующими ключевыми параметрами:

Режим установки

OLM предоставляет три режима установки:

- **Single Namespace**
- **Multi Namespace**

- **Cluster**

Рекомендуется режим Cluster (AllNamespaces). Платформа в будущем будет обновлена до OLM v1, который поддерживает только режим установки AllNamespaces. Поэтому режимы SingleNamespace и MultiNamespace следует избегать.

Канал обновлений

Если Оператор предоставляет несколько каналов обновлений, можно выбрать, на какой канал подписаться, например, **stable**.

Стратегия одобрения

Варианты: **Automatic** или **Manual**.

- **Automatic**: OLM автоматически обновит Оператора при выпуске новой версии в выбранном канале.
- **Manual**: При наличии новой версии OLM создаёт запрос на обновление, который должен быть вручную одобрен администратором кластера перед выполнением обновления.

Примечание: Операторы от Alauda поддерживают только режим **Manual**; в противном случае установка завершится ошибкой.

Место установки

Рекомендуется создавать отдельный namespace для каждого Оператора.

Если несколько Операторов используют один namespace, их Subscriptions могут быть объединены в один InstallPlan:

- Если InstallPlan в этом namespace требует ручного одобрения и находится в ожидании, это может блокировать автоматические обновления других Subscriptions, включённых в тот же InstallPlan.

Установка через веб-консоль

1. Войдите в веб-консоль и переключитесь в режим **Administrator**.
2. Перейдите в **Marketplace > OperatorHub**.
3. Если статус **Absent**:
   - Скачайте пакет Оператора из Alauda Customer Portal или обратитесь в поддержку.
   - Загрузите пакет в целевой кластер с помощью `violet` (см. CLI).
   - На странице **Marketplace > Upload Packages** переключитесь на вкладку **Operator** и подтвердите загрузку.
4. Если статус **Ready**, нажмите **Install** и следуйте руководству пользователя Оператора.

Установка через YAML

Ниже приведены примеры установки Операторов от Alauda (только Manual) и из других источников (Manual или Automatic).

> **INFO**
>
> В отличие от плагинов кластера (которые всегда должны устанавливаться в **глобальный кластер** при использовании YAML), Операторы устанавливаются в **целевой кластер**, где они должны работать. Убедитесь, что вы подключены к нужному кластеру перед выполнением YAML-манифестов.

Ручной режим

`harbor-ce-operator` — оператор от Alauda, поддерживающий только **Manual** одобрение.
В ручном режиме, даже при выпуске новой версии, Оператор не обновится автоматически. Необходимо вручную **одобрить** обновление, чтобы OLM выполнил апгрейд.

1. Проверка доступных версий

```
(
  echo -e "CHANNEL\tNAME\tVERSION"
  kubectl get packagemanifest harbor-ce-operator -o json | jq -r '
    .status.channels[] |
    .name as $channel |
    .entries[] |
    [$channel, .name, .version] | @tsv
  '
) | column -t -s $'\t'
```

Пример вывода:

```
CHANNEL   NAME                          VERSION
harbor-2  harbor-ce-operator.v2.12.11  2.12.11
harbor-2  harbor-ce-operator.v2.12.10  2.12.10
stable    harbor-ce-operator.v2.12.11  2.12.11
stable    harbor-ce-operator.v2.12.10  2.12.10
```

Пояснения к полям:

- **CHANNEL**: имя канала Оператора
- **NAME**: имя ресурса CSV
- **VERSION**: версия Оператора

2. Подтверждение catalogSource

```
kubectl get packagemanifests harbor-ce-operator -ojsonpath='{.status.cata
logSource}'
```

Пример вывода:

```
platform
```

Это означает, что `harbor-ce-operator` поступает из catalogSource `platform`.

3. Создание namespace

```
kubectl create namespace harbor-ce-operator
```

4. Создание Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  annotations:
    cpaas.io/target-namespaces: ""
  name: harbor-ce-operator-subs
  namespace: harbor-ce-operator
spec:
  channel: stable
  installPlanApproval: Manual
  name: harbor-ce-operator
  source: platform
  sourceNamespace: cpaas-system
  startingCSV: harbor-ce-operator.v2.12.11
```

Пояснения к полям:

- **аннотация** `cpaas.io/target-namespaces`: рекомендуется оставить пустой; пустое значение означает установку на весь кластер.
- **.metadata.name**: имя Subscription (должно соответствовать DNS, максимум 253 символа).
- **.metadata.namespace**: namespace для установки Оператора.
- **.spec.channel**: канал подписки Оператора.
- **.spec.installPlanApproval**: стратегия одобрения (`Manual` или `Automatic`). Здесь `Manual` требует ручного одобрения установки/обновления.
- **.spec.source**: catalogSource Оператора.
- **.spec.sourceNamespace**: должно быть `cpaas-system` , так как все catalogSource, предоставляемые платформой, находятся в этом namespace.
- **.spec.startingCSV**: версия для установки при ручном одобрении; если пусто, по умолчанию берётся последняя версия в канале. Не требуется для Automatic.

5. Проверка статуса Subscription

```
kubectl -n harbor-ce-operator get subscriptions harbor-ce-operator-subs -
o yaml
```

Ключевые поля:

- **.status.state**: `UpgradePending` — Оператор ожидает установки или обновления.
- **Condition InstallPlanPending = True**: ожидание ручного одобрения.
- **.status.currentCSV**: текущий подписанный CSV.
- **.status.installPlanRef**: связанный InstallPlan, который должен быть одобрен перед установкой.

6. Одобрение InstallPlan

```
kubectl -n harbor-ce-operator get installplan \
  "$(kubectl -n harbor-ce-operator get subscriptions harbor-ce-operator-s
ubs -o jsonpath='{.status.installPlanRef.name}')"
```

Пример вывода:

```
NAME            CSV                           APPROVAL   APPROVED
install-27t29   harbor-ce-operator.v2.12.11   Manual     false
```

Одобрение вручную:

```
PLAN="$(kubectl -n harbor-ce-operator get subscription harbor-ce-operator
-subs -o jsonpath='{.status.installPlanRef.name}')"
kubectl -n harbor-ce-operator patch installplan "$PLAN" --type=json -p
='[{"op": "replace", "path": "/spec/approved", "value": true}]'
```

Дождитесь создания CSV; фаза изменится на `Succeeded`:

```
kubectl -n harbor-ce-operator get csv
```

Пример вывода:

```
NAME                              DISPLAY                   VERSION   REPLACES
PHASE
harbor-ce-operator.v2.12.11   Alauda Build of Harbor   2.12.11   harbor-c
e-operator.v2.12.10   Succeeded
```

Пояснения к полям:

- **NAME**: имя установленного CSV
- **DISPLAY**: отображаемое имя Оператора
- **VERSION**: версия Оператора
- **REPLACES**: CSV, заменённый при обновлении
- **PHASE**: статус установки (`Succeeded` — успешно)

Автоматический режим

`clickhouse-operator` поступает из источника, не относящегося к Alauda, и его стратегия одобрения может быть установлена в **Automatic**.
В автоматическом режиме Оператор обновляется автоматически при выпуске новой версии без ручного одобрения.

1. Проверка доступных версий

```
(
  echo -e "CHANNEL\tNAME\tVERSION"
  kubectl get packagemanifest clickhouse-operator -o json | jq -r '
    .status.channels[] |
    .name as $channel |
    .entries[] |
    [$channel, .name, .version] | @tsv
  '
) | column -t -s $'\t'
```

Пример вывода:

```
CHANNEL   NAME                              VERSION
stable    clickhouse-operator.v0.18.2       0.18.2
```

2. Подтверждение catalogSource

```
kubectl get packagemanifests clickhouse-operator -ojsonpath='{.status.catalogSource}'
```

Пример вывода:

```
platform
```

Это означает, что `clickhouse-operator` поступает из catalogSource `platform`.

3. Создание namespace

```
kubectl create namespace clickhouse-operator
```

4. Создание Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  annotations:
    cpaas.io/target-namespaces: ""
  name: clickhouse-operator-subs
  namespace: clickhouse-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: clickhouse-operator
  source: platform
  sourceNamespace: cpaas-system
```

Пояснения к полям такие же, как в разделе Ручной режим.

5. Проверка статуса Subscription

```
kubectl -n clickhouse-operator get subscriptions clickhouse-operator-subs
-oyaml
```

6. Проверка CSV

```
kubectl -n clickhouse-operator get csv
```

Пример вывода:

```
NAME                              DISPLAY                VERSION   PHASE
clickhouse-operator.v0.18.2       ClickHouse Operator    0.18.2    Succeeded
```

Установка прошла успешно.

Процесс обновления

Процесс обновления начинается с загрузки новой версии **Оператора**.

После завершения загрузки подождите примерно **10–15 минут**, чтобы платформа синхронизировала информацию о новой версии.

После завершения синхронизации обновления выполняются согласно стратегии, настроенной в **Subscription**:

- Если стратегия одобрения Оператора установлена в **Automatic**, обновление происходит автоматически.
- Если стратегия установлена в **Manual**, запрос на обновление должен быть одобрен вручную. Можно выбрать один из следующих способов обновления:
  - **Пакетное обновление**: выполнить обновление на странице **Platform Management > Cluster Management > Cluster > Features**.
  - **Индивидуальное обновление**: вручную одобрять запросы на обновление в **OperatorHub**.

Примечание: Пакетное обновление поддерживается только для Операторов, предоставляемых Alauda.

Cluster Plugin

Содержание

Overview

Плагин кластера — это инструмент для расширения функциональности платформы. Каждый плагин управляется через три CRD на уровне кластера: **ModulePlugin**, **ModuleConfig** и **ModuleInfo**.

- **ModulePlugin**: Определяет базовую информацию о плагине кластера.

- **ModuleConfig**: Определяет информацию о версии плагина. Каждый ModulePlugin может соответствовать одному или нескольким ModuleConfig.
- **ModuleInfo**: Записывает информацию об установленной версии плагина и его статусе.

Плагины кластера поддерживают динамическую конфигурацию форм. Динамические формы — это простые UI-формы, предоставляющие настраиваемые параметры конфигурации или комбинации параметров для плагинов. Например, при установке Alauda Container Platform Log Collector можно выбрать плагин хранения логов ElasticSearch или ClickHouse через динамическую форму. Определение динамической формы находится в поле `.spec.config` ModuleConfig; если плагину динамическая форма не требуется, это поле пустое.

Плагины публикуются с помощью инструмента **violet**. Обратите внимание:

- Плагины можно публиковать только в **глобальный кластер**, но устанавливать их можно как в глобальном, так и в рабочем кластере в зависимости от конфигурации.
- В одном кластере плагин может быть установлен только один раз.
- После успешной публикации платформа автоматически создаст соответствующие ресурсы ModulePlugin и ModuleConfig в глобальном кластере — ручные изменения не требуются.
- Создание ресурса ModuleInfo устанавливает плагин и позволяет выбрать версию, целевой кластер и параметры динамической формы. Определение динамической формы смотрите в ModuleConfig выбранной версии. Для подробных инструкций по использованию обращайтесь к документации конкретного плагина.

Просмотр доступных плагинов

Чтобы просмотреть все плагины, предоставляемые платформой:

1. Перейдите в представление управления платформой.
2. В левом навигационном меню выберите: **Administrator** > **Marketplace** > **Cluster Plugin**

На этой странице отображаются все доступные плагины и их текущий статус.

Установка через веб-консоль

Если у плагина статус «absent», выполните следующие шаги для его установки:

1. Скачайте пакет плагина:
   - Перейдите в Alauda Customer Portal и скачайте соответствующий пакет плагина.
   - Если у вас нет доступа к Alauda Customer Portal, обратитесь в техническую поддержку.
2. Загрузите пакет на платформу:
   - Используйте инструмент `violet` для публикации пакета на платформе.
   - Подробные инструкции по использованию инструмента смотрите в разделе CLI.
3. Проверьте загрузку:
   - Перейдите в **Administrator** > **Marketplace** > **Upload Packages**
   - Переключитесь на вкладку **Cluster Plugin**
   - Найдите имя загруженного плагина
   - В деталях плагина будут отображены версии загруженного пакета
4. Установите плагин:
   - Если у плагина статус «ready», нажмите **Install**
   - Некоторые плагины требуют параметров установки; смотрите документацию конкретного плагина
   - Плагины без параметров установки начнут установку сразу после нажатия Install

Установка через YAML

Метод установки зависит от типа плагина:

- **Non-config plugin**: Не требует дополнительных параметров; установка простая.
- **Config plugin**: Требует заполнения параметров конфигурации; подробности смотрите в документации плагина.

INFO

Установка через YAML **всегда должна выполняться в глобальном кластере**.

Хотя сам плагин может быть нацелен как на глобальный, так и на рабочий кластер (в зависимости от настроек affinity в ModuleConfig), ресурс `ModuleInfo` можно создавать только в **глобальном кластере**.

Ниже приведены примеры установки через YAML.

non-config

Пример: Alauda Container Platform Web Terminal

1. Проверка доступных версий

Убедитесь, что плагин опубликован, проверив наличие ресурсов ModulePlugin и ModuleConfig в **глобальном кластере**:

```
# kubectl get moduleplugins web-cli
NAME      AGE
web-cli   4d20h

# kubectl get moduleconfigs -l cpaas.io/module-name=web-cli
NAME             AGE
web-cli-v4.0.4   4d21h
```

Это означает, что ModulePlugin `web-cli` существует в глобальном кластере, а версия `v4.0.4` опубликована.

Проверьте ModuleConfig для версии v4.0.4:

```
# kubectl get moduleconfigs web-cli-v4.0.4 -oyaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleConfig
metadata:
  ...
  name: web-cli-v4.0.4
spec:
  affinity:
    clusterAffinity:
      matchLabels:
        is-global: "true"
  version: v4.0.4
  config: {}
  ...
```

Поле `.spec.affinity` определяет affinity к кластеру, указывая, что `web-cli` может быть установлен только в глобальном кластере. `.spec.config` пустое, значит плагин не требует конфигурации и может быть установлен напрямую.

2. Создание ModuleInfo

Создайте ресурс ModuleInfo в глобальном кластере для установки плагина без параметров конфигурации:

```
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: web-cli
    cpaas.io/module-type: plugin
  name: global-temporary-name
spec:
  config: {}
  version: v4.0.4
```

Объяснение полей:

- `name`: Временное имя плагина кластера. Платформа переименует его после создания на основе содержимого в формате `<cluster-name>-<hash содержимого>`, например, `global-ee98c9991ea1464aaa8054bdacbab313`.
- `label cpaas.io/cluster-name`: Указывает целевой кластер для установки плагина. Если конфликтует с affinity из ModuleConfig, установка завершится ошибкой.
  Важно: Эта метка не меняет место применения YAML — YAML **всё равно должен применяться в глобальном кластере**.
- `label cpaas.io/module-name`: Имя плагина, должно совпадать с ресурсом ModulePlugin.
- `label cpaas.io/module-type`: Фиксированное поле, должно быть `plugin`; отсутствие этого поля приведёт к ошибке установки.
- `.spec.config`: Если соответствующий ModuleConfig пуст, это поле можно оставить пустым.
- `.spec.version`: Указывает версию плагина для установки, должна совпадать с `.spec.version` в ModuleConfig.

3. Проверка установки

Так как имя ModuleInfo меняется после создания, найдите ресурс по метке в глобальном кластере, чтобы проверить статус и версию плагина:

```
kubectl get moduleinfo -l cpaas.io/module-name=web-cli
NAME                                     CLUSTER   MODULE    DISPLAY_NAM
E   STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSION
global-ee98c9991ea1464aaa8054bdacbab313   global    web-cli   web-cli
Running   v4.0.4           v4.0.4            v4.0.4
```

Объяснение полей:

- `NAME`: Имя ресурса ModuleInfo
- `CLUSTER`: Кластер, где установлен плагин
- `MODULE`: Имя плагина
- `DISPLAY_NAME`: Отображаемое имя плагина
- `STATUS`: Статус установки; `Running` означает успешную установку и работу

- `TARGET_VERSION`: Целевая версия установки
- `CURRENT_VERSION`: Версия до установки
- `NEW_VERSION`: Последняя доступная версия для установки

with-config

Пример: Alauda Container Platform GPU Device Plugin

1. Проверка доступных версий

Убедитесь, что плагин опубликован, проверив наличие ресурсов ModulePlugin и ModuleConfig в **глобальном кластере**:

```
# kubectl get moduleplugins gpu-device-plugin
NAME                AGE
gpu-device-plugin   4d23h

# kubectl get moduleconfigs -l cpaas.io/module-name=gpu-device-plugin
NAME                          AGE
gpu-device-plugin-v4.0.15   4d23h
```

Это означает, что ModulePlugin `gpu-device-plugin` существует в глобальном кластере, а версия `v4.0.15` опубликована.

Проверьте ModuleConfig для версии v4.0.15:

```
# kubectl get moduleconfigs gpu-device-plugin-v4.0.15 -oyaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleConfig
metadata:
  ...
  name: gpu-device-plugin-v4.0.15
spec:
  affinity:
    clusterAffinity:
      matchExpressions:
      - key: cpaas.io/os-linux
        operator: Exists
      matchLabels:
        cpaas.io/arch-amd64: "true"
  config:
    custom:
      mps_enable: false
      pgpu_enable: false
      vgpu_enable: false
  version: v4.0.15
  ...
```

Примечания:

- Этот плагин можно установить только на кластеры с ОС Linux и архитектурой amd64.
- Динамическая форма содержит три переключателя драйверов устройств: `custom.mps_enable`, `custom.pgpu_enable` и `custom.vgpu_enable`. Только при значении `true` соответствующий драйвер будет установлен.

2. Создание ModuleInfo

Создайте ресурс ModuleInfo **в глобальном кластере** для установки плагина, заполнив параметры динамической формы по необходимости (например, включив драйверы pgpu и vgpu):

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  labels:
    cpaas.io/cluster-name: business
    cpaas.io/module-name: gpu-device-plugin
    cpaas.io/module-type: plugin
  name: business-temporary-name
spec:
  config:
    custom:
      mps_enable: false
      pgpu_enable: true
      vgpu_enable: true
  version: v4.0.15
```

Объяснение полей такое же, как для non-config. Подробности конфигурации смотрите в документации плагина.

3. Проверка установки

Найдите ModuleInfo по метке в глобальном кластере, чтобы проверить статус и версию:

```
# kubectl get moduleinfo -l cpaas.io/module-name=gpu-device-plugin
NAME                                        CLUSTER   MODULE              D
ISPLAY_NAME        STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSI
ON
business-7ebb241b4f77471235e57dd1ec7fbd0d business  gpu-device-plugin   g
pu-device-plugin   Running   v4.0.15          v4.0.15           v4.0.15
```

Объяснение полей такое же, как для non-config.

Процесс обновления

Чтобы обновить существующий плагин до новой версии:

1. Загрузите новую версию:

- Выполните тот же процесс загрузки новой версии на платформу.
- После завершения загрузки подождите примерно **10–15 минут**, чтобы платформа синхронизировала информацию о новой версии.

2. Проверьте новую версию:

- Перейдите в **Administrator** > **Marketplace** > **Upload Packages**
- Переключитесь на вкладку **Cluster Plugin**
- В деталях плагина будет отображена недавно загруженная версия

3. Выполните обновление:

- Перейдите в **Administrator** > **Clusters** > **Clusters**
- Кластеры с доступными обновлениями плагинов будут отображать иконку обновления
- Войдите в детали кластера и переключитесь на вкладку **Features**
- Кнопка обновления будет доступна в компоненте features
- Нажмите **Upgrade** для завершения обновления плагина

Chart Repository

Для получения информации о репозиториях Chart и Helm charts смотрите Working with Helm Charts.

Загрузка пакетов

Платформа предоставляет инструмент командной строки `violet`, который используется для загрузки пакетов с платформы.

Содержание

Загрузка инструмента

Войдите в Alauda Customer Portal, перейдите на страницу **Downloads** и нажмите **CLI Tools**. Скачайте бинарный файл, соответствующий вашей операционной системе и архитектуре.

После загрузки установите инструмент на ваш сервер или ПК.

Для Linux или macOS

Для пользователей без прав root:

```
# Linux x86
sudo mv -f violet_linux_amd64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# Linux ARM
sudo mv -f violet_linux_arm64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# macOS x86
sudo mv -f violet_darwin_amd64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# macOS ARM
sudo mv -f violet_darwin_arm64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
```

Для пользователей с правами root:

```
# Linux x86
mv -f violet_linux_amd64 /usr/bin/violet && chmod +x /usr/bin/violet
# Linux ARM
mv -f violet_linux_arm64 /usr/bin/violet && chmod +x /usr/bin/violet
# macOS x86
mv -f violet_darwin_amd64 /usr/bin/violet && chmod +x /usr/bin/violet
# macOS ARM
mv -f violet_darwin_arm64 /usr/bin/violet && chmod +x /usr/bin/violet
```

Для Windows

1. Скачайте файл и переименуйте его в `violet.exe`, либо используйте PowerShell для переименования:

```
# Windows x86
mv -Force violet_windows_amd64.exe violet.exe
```

2. Запустите инструмент в PowerShell.

> **Примечание**: Если путь к инструменту не добавлен в переменные окружения, при выполнении команд необходимо указывать полный путь.

Требования

Требования к разрешениям

- Необходимо предоставить действующую учетную запись пользователя платформы (аккаунт, имя пользователя и пароль).

Использование

violet ac login

Перед загрузкой пакетов выполните команду `violet ac login` для входа на платформу.

```
violet ac login --account=<account> --username=<username> --password=<pas
sword> --ac-url=<url>
# или используйте существующий токен:
violet ac login --access-token=<token> --ac-url=<url>
```

Дополнительные параметры

```
--account         аккаунт / имя арендатора
--username        имя пользователя
--password        пароль
--ac-url          URL системы AC (по умолчанию: `https://cloud.alauda.io
`)
--access-token    напрямую указать токен доступа, чтобы пропустить вход п
о имени пользователя и паролю
```

> **Note**
>
> Вы можете экспортировать токен доступа из раздела `Customer Portal - Settings`. Срок действия токена доступа — 24 часа с момента успешного входа.

violet ac scenarios

Список доступных сценариев

Вывод: Отформатированная таблица с колонками `ID`, `Name` и `Description`.

```
violet ac scenarios --arch=amd64 --platformVersion=v4.1 --upgrade=true
```

Дополнительные параметры

```
--arch               целевая архитектура (`amd64` , `arm64` , `hybrid`, п
о умолчанию: `amd64`)
--platformVersion    целевая версия платформы
--upgrade            булевый флаг для фильтрации сценариев, связанных с о
бновлением (по умолчанию: `false`)
```

> **Примечание**: Флаг `--upgrade` требуется только при обновлении с ACP 3.x до ACP 4.x. Для ACP 4.x и выше этот флаг не обязателен.

violet ac packages

Список доступных пакетов.

Вывод: Отформатированная таблица с колонками `APP ID`, `APP Name`, `Channel And Version` и `Package`.

```
# загрузить все пакеты для указанной архитектуры, версии платформы и сценария
violet ac packages --arch=<arch> --platformVersion=<version> --scenario=<scenario>
# загрузить все пакеты для указанной архитектуры и версии платформы
violet ac packages --arch=<arch> --platformVersion=<version>
# загрузить один пакет для конкретного ID приложения
violet ac packages --appID=my-app
```

Дополнительные параметры

```
--arch              целевая архитектура (`amd64` , `arm64` , `hybrid`, по умолчанию: `amd64`)
--platformVersion   целевая версия платформы
--appID             загрузить один пакет для конкретного ID приложения
--scenario          фильтр по сценарию (необязательно)
```

> **Note**
>
> Если `scenario` не указан, отображаются все пакеты.

violet ac download-pkg

Загрузка пакетов и их файлов подписи для конкретной архитектуры и версии платформы.

```
violet ac download-pkg --arch=x86 --platformVersion=v4.1.0 --type=core
```

Дополнительные параметры

```
--arch                целевая архитектура (`x86` , `arm` , `hybrid`, по ум
олчанию: `x86`)
--platformVersion     целевая версия платформы
--type                тип пакета (`core`, `extensions`, `standard`, по умо
лчанию: `core`)
```

> **Примечание**: По поводу `type`, для версий v4.0.5 и выше инструмент по умолчанию загружает core-пакет — указывать эту опцию не нужно.
> Для версий с v4.0.0 по v4.0.4 по умолчанию загружается core-пакет; можно указать `extensions` для загрузки расширений.

violet ac download-app

Загрузка пакетов приложений по сценарию (пакетно) или по указанию `appID` и `appVersion`. Команда получает URL для загрузки, затем скачивает пакет и файлы контрольных сумм.

```
violet ac download-app --arch=amd64 --platformVersion=<version> --scenari
o=<scenario>
# Просмотр статуса доступных пакетов
violet ac download-app --arch=amd64 --platformVersion=<version> --scenari
o=<scenario> --check=true
# или загрузка конкретной версии приложения
violet ac download-app --arch=amd64 --appID=<app-id> --appVersion=<app-ve
rsion1>,<app-version2>
```

Дополнительные параметры

```
--arch              целевая архитектура (`amd64` , `arm64` , `hybrid`, п
о умолчанию: `amd64`)
--platformVersion   целевая версия платформы
--appID             конкретный ID приложения
--appVersion        версия приложения (поддерживается несколько версий ч
ерез запятую)
--check             если true, не загружать, а только проверить статус д
оступных пакетов (по умолчанию: `false`)
--scenario          имя сценария (необязательно, загрузка последних верс
ий для сценария)
```

> **Note**
>
> Если `scenario` не указан, загружаются все пакеты.

violet ac import-yaml

Чтение локального YAML-файла (по умолчанию `./apps.yaml`, экспорт из `violet list`), содержащего карту `applications`, отправка его в endpoint проверки сценария AC и отображение результатов валидации. Опционально загрузка проверенных пакетов.

```
violet ac import-yaml --arch=amd64 --platformVersion=v4.1.3 --download=tr
ue
```

Дополнительные параметры

```
--arch              целевая архитектура (`amd64` , `arm64` , `hybrid`, п
о умолчанию: `amd64`)
--platformVersion   целевая версия платформы
--download          булево значение; если true, попытаться загрузить про
веренные пакеты после проверки
--path              путь к YAML-файлу (по умолчанию `./apps.yaml`)
```

Примеры рабочих процессов

- Вход и сохранение токена:

```
violet ac login --account=tenantA --username=admin --password=password
--ac-url=https://ac.example.com
```

- Список доступных сценариев:

```
violet ac scenarios --arch=amd64 --platformVersion=v4.1
```

- Загрузка core-пакетов целевой версии:

```
violet ac download-pkg --arch=x86 --platformVersion=v4.1.0
```

- Список доступных пакетов и загрузка последней версии для сценария:

```
violet ac download-app --arch=amd64 --platformVersion=v4.1.0 --scenario
=my-scenario
```

- Просмотр статуса пакетов ('Downloaded' или 'Download Failed'):

```
violet ac download-app --arch=amd64 --platformVersion=v4.1.0 --scenario
=my-scenario --check=true
```

- Валидация YAML приложений и загрузка проверенных пакетов:

```
violet ac import-yaml --arch=amd64 --platformVersion=v4.1.0 --download=
true --path=./apps.yaml
```

Upload Packages

Платформа предоставляет инструмент командной строки `violet`, который используется для загрузки пакетов, скачанных из Marketplace в Alauda Customer Portal, на платформу.

`violet` поддерживает загрузку следующих типов пакетов:

- **Operator**
- **Cluster Plugin**
- **Helm Chart**

Если статус пакета в **Cluster Plugins** или **OperatorHub** отображается как `Absent`, необходимо использовать этот инструмент для загрузки соответствующего пакета.

Процесс загрузки с помощью `violet` включает следующие основные шаги:

1. Распаковка и получение информации из пакета
2. Отправка образов в реестр образов
3. Создание ресурсов **Artifact** и **ArtifactVersion** на платформе

Содержание

Загрузка инструмента

Войдите в Alauda Customer Portal, перейдите на страницу **Downloads** и выберите **CLI Tools**. Скачайте бинарный файл, соответствующий вашей операционной системе и архитектуре.

После загрузки установите инструмент на ваш сервер или ПК.

Для Linux или macOS

Для пользователей без root-прав:

```
# Linux x86
sudo mv -f violet_linux_amd64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# Linux ARM
sudo mv -f violet_linux_arm64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# macOS x86
sudo mv -f violet_darwin_amd64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
# macOS ARM
sudo mv -f violet_darwin_arm64 /usr/local/bin/violet && sudo chmod +x /us
r/local/bin/violet
```

Для пользователей с root-правами:

```
# Linux x86
mv -f violet_linux_amd64 /usr/bin/violet && chmod +x /usr/bin/violet
# Linux ARM
mv -f violet_linux_arm64 /usr/bin/violet && chmod +x /usr/bin/violet
# macOS x86
mv -f violet_darwin_amd64 /usr/bin/violet && chmod +x /usr/bin/violet
# macOS ARM
mv -f violet_darwin_arm64 /usr/bin/violet && chmod +x /usr/bin/violet
```

Для Windows

1. Скачайте файл и переименуйте его в `violet.exe`, либо используйте PowerShell для переименования:

   ```
   # Windows x86
   mv -Force violet_windows_amd64.exe violet.exe
   ```

2. Запустите инструмент в PowerShell.

> **Примечание**: Если путь к инструменту не добавлен в переменные окружения, при выполнении команд необходимо указывать полный путь.

Требования

Требования к правам

- Необходимо предоставить действующую учетную запись пользователя платформы (имя пользователя и пароль).
- Учетная запись должна иметь свойство роли, установленное в `System`, а имя роли должно быть `platform-admin-system`.

> **Примечание:** Если свойство роли вашей учетной записи установлено в `Custom`, вы не сможете использовать этот инструмент.

Использование

Общие параметры

Некоторые команды `violet` принимают следующие параметры. Для конкретного использования смотрите разделы отдельных команд.

Параметры подключения к платформе

```
--platform-address <URL доступа к платформе>     # URL доступа к платформ
е, например, "https://example.com"
--platform-username <пользователь платформы>    # Имя пользователя платфо
рмы
--platform-password <пароль пользователя>       # Пароль пользователя пла
тформы
```

Параметры реестра образов

```
--dest-repo <адрес репозитория образов>         # Указать адрес целевого
репозитория образов, например, "harbor.demo.io"
--username <пользователь реестра>                # Имя пользователя указа
нного реестра образов
--password <пароль реестра>                      # Пароль указанного реес
тра образов
--no-auth                                        # Указать, если реестр о
бразов не требует аутентификации
--plain                                          # Указать, если реестр о
бразов использует HTTP вместо HTTPS
```

WARNING

Ограничение на IPv6-адреса

Для параметров `--platform-address` и `--dest-repo`:

- Если используется IP-адрес (а не доменное имя), **IPv6 формат НЕ поддерживается**
- Поддерживаются только IPv4-адреса или доменные имена

violet show

Перед загрузкой пакета используйте команду `violet show` для предварительного просмотра его деталей.

```
violet show topolvm-operator.v2.3.0.tgz
Name: NativeStor
Type: bundle
Arch: [linux/amd64]
Version: 2.3.0

violet show topolvm-operator.v2.3.0.tgz --all
Name: NativeStor
Type: bundle
Arch: []
Version: 2.3.0
Artifact: harbor.demo.io/acp/topolvm-operator-bundle:v3.11.0
RelateImages: [harbor.demo.io/acp/topolvm-operator:v3.11.0 harbor.demo.i
o/acp/topolvm:v3.11.0 harbor.demo.io/3rdparty/k8scsi/csi-provisioner:v3.0
0 ...]
```

violet list

При обновлении платформы вы можете вывести список всех плагинов, загруженных на платформу, и экспортировать результат в файл. Сгенерированный файл затем можно загрузить в Alauda Cloud для скачивания необходимых пакетов плагинов.

Необязательные флаги

```
--output-file <файл вывода>                    # Путь к файлу списка пл
агинов для вывода
```

Для параметров подключения к платформе (`--platform-address`, `--platform-username`, `--platform-password`) смотрите Общие параметры.

violet verify

Команда `violet verify` используется для проверки подписи одного или нескольких пакетов перед их загрузкой. Поддерживаются два метода проверки: **контрольная сумма** и **GPG**. Пакет (`.tgz`) и соответствующий файл подписи должны находиться в одной директории.

```
violet verify example.tgz
# или проверить все пакеты в директории
violet verify packages_dir_name
```

Пример вывода:

```
verify path: /path/to/packages
====== Verification Summary ======
Verified successfully with GPG: 2 file(s)
  - /path/to/packages/redis-operator.tgz
  - /path/to/packages/mysql-operator.tgz

Verified successfully with checksum: 1 file(s)
  - /path/to/packages/nginx-controller.tgz

Verification failed: 1 file(s)
  - /path/to/packages/etcd-operator.tgz

No verification file found: 1 file(s)
  - /path/to/packages/demo-plugin.tgz
```

Объяснение:

- **Verified successfully with GPG** — перечисленные файлы успешно проверены с помощью **GPG-файлов подписи** (с расширением `.sig`).
- **Verified successfully with checksum** — файлы, проверенные с помощью файлов контрольных сумм (например, `.sha256`), прошли проверку целостности.
- **Verification failed** — перечисленные файлы не прошли проверку из-за несоответствия или недействительных подписей.
- **No verification file found** — в директории не найден соответствующий файл `.sig` (GPG) или файл контрольной суммы.

Необязательные флаги

```
--debug       Использовать уровень логирования debug.
-h, --help    Показать справочную информацию по команде verify.
```

violet push

Ниже приведены примеры типичных сценариев использования.

Для параметров подключения к платформе и реестру образов смотрите Общие параметры.

Необязательные флаги

```
--clusters <имена кластеров>                    # Указать целевые кластер
ы через запятую (например, region1,region2)
```

При указании `--dest-repo` обязательно необходимо предоставить либо данные аутентификации реестра образов, либо параметр `--no-auth`.

Загрузка Operator в несколько кластеров

```
violet push opensearch-operator.v3.14.2.tgz \
  --platform-address "https://example.com" \
  --platform-username "<platform_user>" \
  --platform-password "<platform_password>" \
  --clusters region1,region2
```

INFO

- Если параметр `--clusters` не указан, Operator загружается по умолчанию в **глобальный кластер**.\

Загрузка Operator в резервный глобальный кластер

```
violet push opensearch-operator.v3.14.2.tgz \
  --platform-address "https://<standby-platform-address>" \
  --platform-username "<platform_user>" \
  --platform-password "<platform_password>" \
  --dest-repo "<standby-cluster-VIP>:11443" --username "<registry-usernam
e>" --password "<registry-password>"
```

WARNING

При использовании `violet` для загрузки пакетов в резервный кластер:

- Параметр `--dest-repo <VIP адрес резервного кластера>` **ДОЛЖЕН** быть указан
- Параметр `--platform-address` **ДОЛЖЕН** указывать на адрес доступа платформы **резервного кластера**
- Необходимо предоставить либо данные аутентификации реестра образов резервного кластера, либо параметр `--no-auth`

В противном случае пакеты будут загружены в репозиторий образов **основного кластера**, что помешает резервному кластеру устанавливать или обновлять расширения.

Загрузка Cluster Plugin

```
violet push plugins-cloudedge-v0.3.16-hybrid.tgz \
  --platform-address "https://example.com" \
  --platform-username "<platform_user>" \
  --platform-password "<platform_password>"
```

INFO

- При загрузке Cluster Plugin параметр `--clusters` указывать не нужно, так как платформа автоматически распределит плагин согласно его конфигурации affinity. Если `--clusters` указан, параметр будет проигнорирован.\

Загрузка Helm Chart в репозиторий чартов

```
violet push plugins-cloudedge-v0.3.16-hybrid.tgz \
  --platform-address "https://example.com" \
  --platform-username "<platform_user>" \
  --platform-password "<platform_password>"
```

INFO

- Helm Charts можно загружать только в репозиторий по умолчанию `public-charts`, предоставляемый платформой.\

Одновременная загрузка всех пакетов

Если из Marketplace скачано несколько пакетов, их можно поместить в одну директорию и загрузить все сразу:

```
violet push <packages_dir_name> \
  --platform-address "https://example.com" \
  --platform-username "<platform_user>" \
  --platform-password "<platform_password>" \
  --clusters "<cluster_name>"
```

WARNING

Если целью обновления является **глобальный кластер**, параметр `--clusters` можно опустить, так как по умолчанию загрузка происходит в глобальный кластер.

Однако если целью обновления является кластер нагрузки, параметр `--clusters <имя_кластера_нагрузки>` **обязателен**.