Menu

# Расширение

## Обзор

## Оператор

## Плагин кластера

## Загрузка пакетов

# 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, с удобным интерфейсом 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

# Overview

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 InstallPlans.
- **CatalogSource**: CRD с областью действия namespace, который управляет источником каталога Операторов и предоставляет метаданные Оператора (например, информацию о версиях, управляемых CRD). Платформа предоставляет 3 стандартных CatalogSource: **system**, **platform** и **custom**. Операторы из **system** не отображаются в OperatorHub.
- **ClusterServiceVersion (CSV)**: CRD с областью действия namespace, описывающий конкретную версию Оператора, включая необходимые ресурсы, CRD и разрешения.
- **Subscription**: CRD с областью действия namespace, описывающий подписанный Оператор, его источник, канал получения и стратегию обновления.
- **InstallPlan**: CRD с областью действия namespace, описывающий фактические операции установки (например, создание Deployments, CRD, RBAC). Оператор будет установлен или обновлен только после утверждения InstallPlan.

# Operator Sources

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 обязательства по SLA.

2. **Curated**
   Отобраны из сообщества с открытым исходным кодом, соответствуют 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 не предоставляются.

4. **Marketplace**
   Предоставляются и поддерживаются сторонними поставщиками, сертифицированными Alauda . Alauda обеспечивает поддержку интеграции с платформой, а поставщик отвечает за основное сопровождение.

5. **Custom**
   Разработаны и загружены пользователем для удовлетворения индивидуальных требований.

---

# Pre-installation Preparation

Перед установкой Оператора необходимо ознакомиться со следующими ключевыми параметрами:

## Installation Mode

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 следует избегать SingleNamespace и MultiNamespace.

## Update Channel

Если Оператор предоставляет несколько каналов обновления, можно выбрать канал для подписки, например, **stable**.

## Approval Strategy

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

- **Automatic**: OLM автоматически обновит Оператора при выпуске новой версии в выбранном канале.
- **Manual**: При появлении 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.

## Installation Location

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.

---

# Installing via Web Console

1. Войдите в веб-консоль и переключитесь в режим **Administrator**.

2. Перейдите в **Marketplace > OperatorHub**.
3. Если статус **Absent**:
   - Скачайте пакет Оператора из Custom 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.

---

# Installing via YAML

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

## Manual

`harbor-ce-operator` — Оператор от Alauda , поддерживает только **Manual** одобрение. В режиме Manual, даже при выпуске новой версии, Оператор не обновится автоматически. Необходимо вручную **Approve** перед выполнением обновления OLM.

### 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.catalogSource}'
```

Пример вывода:

```
platform
```

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

## 3. Создание namespace

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

## 4. Создание Subscription

```yaml
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:

- **annotation** `cpaas.io/target-namespaces`: рекомендуется оставить пустым; 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 Manual одобрении; если пусто, устанавливается последняя версия в канале. Не требуется для Automatic.

## 5. Проверка статуса Subscription

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

Основные поля вывода:

- **.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-subs -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-ce-
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` означает успешную установку)

## Automatic

`clickhouse-operator` — Оператор из не- Alauda источника, стратегия одобрения может быть **Automatic**.
В режиме Automatic 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:

```
community-operators
```

Это означает, что `clickhouse-operator` взят из catalogSource `community-operators`.

## 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: community-operators
  sourceNamespace: openshift-marketplace
```

Пояснения к полям такие же, как в Manual.

### 5. Проверка статуса Subscription

```
kubectl -n clickhouse-operator get subscriptions clickhouse-operator -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.

---

## Upgrade Process

1. Загрузите новую версию Оператора.
2. Обновления выполняются согласно стратегии, настроенной в Subscription:
   - **Automatic Upgrade**: обновление происходит автоматически после загрузки.
   - **Manual Upgrade**:
     - **Пакетное обновление**: выполняется на странице **Platform Management > Cluster Management > Cluster > Features**.
     - **Индивидуальное обновление**: вручную одобрять запросы на обновление в OperatorHub.

**Примечание**: пакетные обновления поддерживаются только для Операторов от 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**: Фиксирует установленную версию плагина и информацию о его статусе.

Плагины кластера поддерживают динамическую конфигурацию форм. Динамические формы — это простые UI-формы, предоставляющие настраиваемые параметры конфигурации или их комбинации 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 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 установки:

1. Скачайте пакет плагина:
   - Перейдите в Custom Portal и скачайте соответствующий пакет плагина.
   - Если у вас нет доступа к Custom Portal, обратитесь в техническую поддержку.
2. Загрузите пакет на платформу:
   - Используйте инструмент `violet` для публикации пакета на платформу.
   - 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**: Дополнительные параметры не требуются; установка простая.

- **Config plugin**: Требуется заполнение параметров конфигурации; подробности в документации плагина.

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:

```
# 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 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 аффинити кластера, указывая, что `web-cli` можно устанавливать только в глобальном кластере. `.spec.config` пустое, значит плагин не требует конфигурации и может быть установлен напрямую.

## 2. Создание ModuleInfo

Создайте ресурс ModuleInfo 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 кластер, в котором должен быть установлен плагин. Если конфликтует с аффинити ModuleInfo, установка завершится ошибкой.
- `label cpaas.io/module-name`: Имя плагина, должно совпадать с ресурсом ModulePlugin.
- `label cpaas.io/module-type`: Фиксированное поле, должно быть `plugin`; отсутствие 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 при создании, найдите ресурс по лейблу для проверки статуса и версии плагина:

```
kubectl get moduleinfo -l cpaas.io/module-name=web-cli
NAME                                CLUSTER   MODULE    DISPLAY_NAME   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 ресурсы ModulePlugin и ModuleConfig:

```
# 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, 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 устанавливать только в кластерах с ОС Linux и архитектурой amd64.
- Динамическая форма включает три переключателя драйверов устройств: `custom.mps_enable`, `custom.pgpu_enable` и `custom.vgpu_enable`. Соответствующий драйвер будет установлен только при значении `true`.

## 2. Создание ModuleInfo

Создайте ресурс ModuleInfo 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 лейблу для проверки статуса и версии:

```
# kubectl get moduleinfo -l cpaas.io/module-name=gpu-device-plugin
NAME                                        CLUSTER   MODULE              DISPLAY_NAME
STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSION
business-7ebb241b4f77471235e57dd1ec7fbd0d business  gpu-device-plugin   gpu-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.

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 для обновления плагинами будут иметь иконку обновления
- Войдите в детали кластера и переключитесь на вкладку **Features**
- Кнопка обновления будет доступна в компоненте features
- Нажмите **Upgrade** для завершения обновления плагина

# Загрузка пакетов

Платформа предоставляет инструмент командной строки `violet`, который используется для загрузки пакетов, скачанных из Marketplace в Custom 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

**Поддерживаемые операционные системы и архитектуры**

- Linux, macOS, Windows
- Для Linux и macOS поддерживаются архитектуры **x86** и **ARM**

**Шаги для загрузки**

1. Войдите в Web Console глобального кластера и переключитесь в режим **Administrator**.
2. Перейдите в раздел **Marketplace > Upload Packages**.
3. Нажмите **Download Packaging and Listing Tool**.
4. Выберите 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:**

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

**Для пользователей 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 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 инструмента

Ниже приведены примеры типичных сценариев использования.

## Просмотр информации о пакете

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.io/acp/topolvm:v3.11.0 harbor.demo.io/3rdparty/k8scsi/csi-provisioner:v3.00
...]
```

## Загрузка Operator в несколько кластеров

Используйте параметр `--clusters` для указания целевых кластеров.

```
violet push opensearch-operator.v3.14.2.tgz \
  --platform-address https://192.168.0.1 \
  --platform-username <user> \
  --platform-password <password> \
  --clusters region1,region2
```

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

## Загрузка Cluster Plugin

```
violet push plugins-cloudedge-v0.3.16-hybrid.tgz \
  --platform-address https://192.168.0.1 \
  --platform-username <user> \
  --platform-password <password>
```

> **Примечание:** При загрузке Cluster Plugin указывать параметр `--clusters` не нужно, так как платформа автоматически распределит его согласно конфигурации affinity. Если параметр `--clusters` указан, он будет проигнорирован.

## Загрузка всех пакетов из каталога

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

```
violet push <packages_dir_name> \
  --platform-address https://192.168.0.1 \
  --platform-username <user> \
  --platform-password <password>
```

Инструмент автоматически определит типы пакетов в каталоге.

## Загрузка Helm Chart

Загрузите Helm Chart в репозиторий чартов:

```
violet push plugins-cloudedge-v0.3.16-hybrid.tgz \
  --platform-address https://192.168.0.1 \
  --platform-username <user> \
  --platform-password <password>
```

> **Примечание:** Helm Charts можно загружать только в репозиторий по умолчанию `public-charts`, предоставляемый платформой.

Для получения дополнительной информации выполните:

```
violet --help
```