

# 扩展

---

概览

**Operator**

集群插件

上架软件包

# 概览

该平台提供了一个全面的扩展系统，允许用户增强其 Kubernetes 集群的功能。该系统设计灵活且易于使用，使用户能够轻松为集群添加新功能和能力。

该系统由两种主要的扩展类型组成：

- **Operators**：Operators 基于 Operator Lifecycle Manager (OLM) v0 框架构建，为平台提供专业的操作能力。这些扩展使您能够自动化管理集群内的复杂应用和服务。
- **集群插件**：平台拥有专门为 Chart 类型插件设计的专有集群插件系统。与标准方法相比，该系统提供了更优的安装和管理体验，并配备了用户友好的界面来处理基于 Chart 的扩展。

通过支持众多 Operators 和集群插件，用户可以显著扩展平台的能力，以满足特定的运营需求和使用场景。

# Operator

---

## 目录

概述

Operator 来源

安装前准备

安装模式

更新通道

审批策略

安装位置

通过 Web 控制台安装

通过 YAML 安装

手动模式

1. 查看可用版本
2. 确认 catalogSource
3. 创建命名空间
4. 创建 Subscription
5. 查看 Subscription 状态
6. 批准 InstallPlan

自动模式

1. 查看可用版本
  2. 确认 catalogSource
  3. 创建命名空间
  4. 创建 Subscription
  5. 查看 Subscription 状态
-

## 6. 验证 CSV

### 升级流程

---

## 概述

基于 **OLM (Operator Lifecycle Manager)** 框架，**OperatorHub** 提供了一个统一的界面，用于管理 Operators 的安装、升级和生命周期。管理员可以使用 OperatorHub 安装和管理 Operators，实现 Kubernetes 应用的全生命周期自动化，包括创建、更新和删除。

OLM 主要由以下组件和 CRD 组成：

- **OLM (olm-operator)**：管理 Operators 的完整生命周期，包括安装、升级和版本冲突检测。
- **Catalog Operator**：管理 Operator 目录并生成相应的 InstallPlans。
- **CatalogSource**：一个命名空间范围的 CRD，管理 Operator 目录源并提供 Operator 元数据（如版本信息、管理的 CRD）。平台提供了 3 个默认的 CatalogSources：**system**、**platform** 和 **custom**。**system** 中的 Operators 不会在 OperatorHub 中显示。
- **ClusterServiceVersion (CSV)**：一个命名空间范围的 CRD，描述 Operator 的特定版本，包括其所需的资源、CRD 和权限。
- **Subscription**：一个命名空间范围的 CRD，描述订阅的 Operator、其来源、获取渠道和升级策略。
- **InstallPlan**：一个命名空间范围的 CRD，描述实际执行的安装操作（如创建 Deployment、CRD、RBAC）。Operator 只有在 InstallPlan 被批准后会安装或升级。

## Operator 来源

为明确 OperatorHub 中不同 Operators 的生命周期策略，平台提供了 5 种来源类型：

### 1. 灵雀云

由灵雀云提供和维护，包括全生命周期管理、安全更新、技术支持和 SLA 承诺。

---

## 2. Curated

从开源社区精选，版本与社区一致，无代码修改或重新编译。灵雀云 提供指导和安全更新，但不保证 SLA 或生命周期管理。

## 3. Community

由开源社区提供，定期更新以保证可安装性，但不保证功能完整性；不提供 SLA 或 灵雀云 支持。

## 4. Marketplace

由通过 灵雀云 认证的第三方厂商提供和维护。灵雀云 提供平台集成支持，核心维护由厂商负责。

## 5. Custom

由用户开发并上传，以满足自定义使用需求。

# 安装前准备

在安装 Operator 之前，需要了解以下关键参数：

## 安装模式

OLM 提供三种安装模式：

- 单命名空间
- 多命名空间
- 集群级

推荐使用集群级模式 (**AllNamespaces**)。平台最终将升级至 OLM v1，仅支持 AllNamespaces 安装模式，因此应尽量避免使用 SingleNamespace 和 MultiNamespace。

## 更新通道

如果 Operator 提供多个更新通道，可以选择订阅其中一个通道，例如 **stable**。

## 审批策略

选项：自动 或 手动。

- 自动：当选定通道发布新版本时，OLM 会自动升级 Operator。
- 手动：当有新版本时，OLM 会创建升级请求，必须由集群管理员手动批准后会升级。

注意：来自 灵雀云 的 Operators 仅支持 手动 模式，否则安装会失败。

## 安装位置

建议为每个 Operator 创建独立的命名空间。

如果多个 Operators 共享同一命名空间，它们的 Subscriptions 可能会被合并为单个 InstallPlan：

- 如果该命名空间中的 InstallPlan 需要手动审批且处于待审批状态，可能会阻塞同一 InstallPlan 中其他 Subscriptions 的自动升级。

## 通过 Web 控制台安装

1. 登录 Web 控制台并切换到 管理员 视图。
2. 进入 **Marketplace > OperatorHub**。
3. 如果状态为 **Absent**：
  - 从 Custom Portal 下载 Operator 软件包或联系支持。
  - 使用 `violet` 将软件包上传至目标集群（详见 [CLI](#)）。
  - 在 **Marketplace > Upload Packages** 页面切换到 **Operator** 标签页，确认上传。
4. 如果状态为 **Ready**，点击 **安装** 并按照 Operator 用户指南操作。

## 通过 YAML 安装

以下示例演示了来自 灵雀云（仅手动模式）和非 灵雀云 来源（手动或自动模式）的 Operator 安装方法。

## INFO

与集群插件（使用 YAML 时必须始终安装在 **global cluster**）不同，Operator 是安装在希望运行它们的目标集群中。执行任何 YAML 清单前，请确保已连接到目标集群。

## 手动模式

`harbor-ce-operator` 来自 灵雀云，仅支持 手动 审批。

手动模式下，即使发布新版本，Operator 也不会自动升级，必须先手动 批准，OLM 才会执行升级。

### 1. 查看可用版本

```
(
  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'
```

示例输出：

```
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
```

字段说明：

- **CHANNEL** : Operator 通道名称
- **NAME** : CSV 资源名称
- **VERSION** : Operator 版本

## 2. 确认 catalogSource

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

示例输出：

```
platform
```

表示 harbor-ce-operator 来自 platform catalogSource。

## 3. 创建命名空间

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

## 4. 创建 Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  annotations:
    cpaas.io/target-namespaces: ""
  name: harbor-ce-operator-sub
  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
```

字段说明：

- **annotation** cpaas.io/target-namespaces：建议设置为空，空表示集群范围安装。
- **.metadata.name**：Subscription 名称（符合 DNS 规范，最长 253 字符）。
- **.metadata.namespace**：Operator 安装的命名空间。

- **.spec.channel** : 订阅的 Operator 通道。
- **.spec.installPlanApproval** : 审批策略 ( `Manual` 或 `Automatic` ) , 此处为 `Manual` , 需手动批准安装/升级。
- **.spec.source** : Operator catalogSource。
- **.spec.sourceNamespace** : 必须设置为 `cpaas-system` , 因为平台提供的所有 catalogSources 均位于该命名空间。
- **.spec.startingCSV** : 指定安装的版本 (手动审批时必填) ; 为空则默认安装通道中最新版本。自动审批时不需要。

## 5. 查看 Subscription 状态

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

关键输出 :

- **.status.state** : `UpgradePending` 表示 Operator 正在等待安装或升级。
- **Condition InstallPlanPending = True** : 等待手动批准。
- **.status.currentCSV** : 当前订阅的最新 CSV。
- **.status.installPlanRef** : 关联的 InstallPlan , 必须批准后才能继续安装。

## 6. 批准 InstallPlan

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

示例输出 :

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

手动批准 :

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

等待 CSV 创建，Phase 变为 `Succeeded`：

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

示例输出：

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

字段说明：

- **NAME**：已安装的 CSV 名称
- **DISPLAY**：Operator 显示名称
- **VERSION**：Operator 版本
- **REPLACES**：升级时被替换的 CSV
- **PHASE**：安装状态（`Succeeded` 表示成功）

## 自动模式

`clickhouse-operator` 来自非灵雀云来源，其审批策略可设置为自动。自动模式下，当发布新版本时，Operator 会自动升级，无需手动批准。

### 1. 查看可用版本

```
(  
  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'
```

示例输出：

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

## 2. 确认 catalogSource

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

示例输出：

```
community-operators
```

表示 `clickhouse-operator` 来自 `community-operators` catalogSource。

## 3. 创建命名空间

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

## 4. 创建 Subscription

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

字段说明同手动模式。

## 5. 查看 Subscription 状态

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

## 6. 验证 CSV

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

示例输出：

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

安装成功。

# 升级流程

1. 上传新的 Operator 版本。

## 2. 升级遵循 Subscription 中配置的策略：

- 自动升级：上传后自动升级。
- 手动升级：
  - 批量升级：在 平台管理 > 集群管理 > 集群 > 功能 页面执行。
  - 单独升级：在 OperatorHub 中手动批准升级请求。

注意：仅 灵雀云 来源的 Operators 支持批量升级。

# Cluster Plugin

---

## 目录

Overview

查看可用插件

通过 Web 控制台安装

通过 YAML 安装

non-config

1. 检查可用版本
2. 创建 ModuleInfo
3. 验证安装

with-config

1. 检查可用版本
2. 创建 ModuleInfo
3. 验证安装

升级流程

---

## Overview

集群插件是用于扩展平台功能的工具。每个插件通过三个集群级别的 CRD 进行管理：**ModulePlugin**、**ModuleConfig** 和 **ModuleInfo**。

- **ModulePlugin**：定义集群插件的基本信息。
-

- **ModuleConfig** : 定义插件的版本信息。每个 ModulePlugin 可以对应一个或多个 ModuleConfig。
- **ModuleInfo** : 记录已安装插件的版本和状态信息。

集群插件支持动态表单配置。动态表单是简单的 UI 表单，为插件提供可自定义的配置选项或参数组合。例如，安装 灵雀云容器平台 日志采集器时，可以通过动态表单选择日志存储插件为 ElasticSearch 或 ClickHouse。动态表单定义位于 ModuleConfig 的 `.spec.config` 字段；如果插件不需要动态表单，该字段为空。

插件通过 **violet** 工具发布。注意：

- 插件只能发布到 **global cluster**，但可根据配置安装到 global cluster 或 workload cluster。
- 同一集群内，插件只能安装一次。
- 发布成功后，平台会自动在 global cluster 创建对应的 ModulePlugin 和 ModuleConfig，无需手动修改。
- 创建 ModuleInfo 资源即可安装插件，并可选择版本、目标集群及动态表单参数。动态表单定义请参考所选版本的 ModuleConfig。更多使用说明请参考插件相关文档。

## 查看可用插件

查看平台提供的所有插件：

1. 进入平台管理视图。
2. 点击左侧导航菜单：管理员 > **Marketplace** > **Cluster Plugin**

该页面列出所有可用插件及其当前状态。

## 通过 Web 控制台安装

如果插件显示“absent”状态，按以下步骤安装：

1. 下载插件包：
  - 访问 Custom Portal 下载对应插件包。

- 若无访问权限，请联系技术支持。

## 2. 上传包至平台：

- 使用 `violet` 工具将包发布到平台。
- 详细使用说明请参考 [CLI](#)。

## 3. 验证上传：

- 进入 管理员 > **Marketplace** > **Upload Packages**
- 切换到 **Cluster Plugin** 标签页
- 找到已上传插件名称
- 插件详情会显示上传包的版本信息

## 4. 安装插件：

- 若插件显示“ready”状态，点击 **Install**
- 部分插件需填写安装参数，详见插件文档
- 无安装参数的插件点击安装后即开始安装

# 通过 YAML 安装

安装方式根据插件类型不同：

- 非配置插件：无需额外参数，安装简单。
- 配置插件：需填写配置参数，详见插件文档。

### INFO

基于 YAML 的安装必须始终在 **global cluster** 中进行。

虽然插件本身可针对 **global cluster** 或 **workload cluster**（取决于 `ModuleConfig` 中的亲和性设置），但 `ModuleInfo` 资源只能在 **global cluster** 中创建。

以下示例演示基于 YAML 的安装。

# non-config

示例：灵雀云容器平台 Web Terminal

## 1. 检查可用版本

确认插件已发布，检查 **global cluster** 中是否存在 **ModulePlugin** 和 **ModuleConfig** 资源：

```
# 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
```

表示 **global cluster** 中存在 **ModulePlugin** `web-cli`，且版本 `v4.0.4` 已发布。

查看版本 `v4.0.4` 的 **ModuleConfig**：

```
# 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: {}
  ...
```

`.spec.affinity` 定义集群亲和性，表明 `web-cli` 只能安装在 **global cluster**。`.spec.config` 为空，表示插件无需配置，可直接安装。

## 2. 创建 **ModuleInfo**

在 global cluster 中创建 ModuleInfo 资源，安装插件且不填写配置参数：

```
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
```

字段说明：

- `name`：集群插件临时名称，平台创建后会根据内容重命名，格式为 `<cluster-name>-<内容哈希>`，例如 `global-ee98c9991ea1464aaa8054bdacbab313`。
- `label cpaas.io/cluster-name`：指定插件安装的目标集群。若与 ModuleConfig 的亲中性冲突，安装会失败。  
  
\*\*注意：\*\*该标签不改变 YAML 应用的集群，YAML 仍必须应用于 **global cluster**。
- `label cpaas.io/module-name`：插件名称，必须与 ModulePlugin 资源匹配。
- `label cpaas.io/module-type`：固定字段，必须为 `plugin`，缺失此字段安装失败。
- `.spec.config`：对应 ModuleConfig 为空时可留空。
- `.spec.version`：指定安装的插件版本，必须与 ModuleConfig 中 `.spec.version` 匹配。

### 3. 验证安装

由于 ModuleInfo 名称创建后会变更，可通过标签在 global cluster 中查找资源，查看插件状态和版本：

```
kubectl get moduleinfo -l cpaas.io/module-name=web-cli
```

NAME	CLUSTER	MODULE	DISPLAY_NAME	STATUS
global-ee98c9991ea1464aaa8054bdacbab313	global	web-cli	web-cli	Running
TARGET_VERSION	CURRENT_VERSION	NEW_VERSION		
v4.0.4	v4.0.4	v4.0.4		

字段说明：

- **NAME** : ModuleInfo 资源名称
- **CLUSTER** : 插件安装的集群
- **MODULE** : 插件名称
- **DISPLAY\_NAME** : 插件显示名称
- **STATUS** : 安装状态，**Running** 表示安装成功且运行中
- **TARGET\_VERSION** : 目标安装版本
- **CURRENT\_VERSION** : 安装前版本
- **NEW\_VERSION** : 可升级的最新版本

## with-config

示例：灵雀云容器平台 GPU Device Plugin

### 1. 检查可用版本

确认插件已发布，检查 **global cluster** 中 ModulePlugin 和 ModuleConfig 资源：

```
# 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

表示 global cluster 中存在 ModulePlugin **gpu-device-plugin**，且版本 **v4.0.15** 已发布。

查看版本 v4.0.15 的 ModuleConfig :

```
# 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
  ...
```

说明 :

- 该插件只能安装在操作系统为 Linux 且架构为 amd64 的集群。
- 动态表单包含三个设备驱动开关：`custom.mps_enable`、`custom.pgpu_enable` 和 `custom.vgpu_enable`，仅设置为 `true` 时对应驱动才会安装。

## 2. 创建 ModuleInfo

在 **global cluster** 中创建 ModuleInfo 资源，填写动态表单参数（例如启用 pgpu 和 vgpu 驱动）安装插件：

```

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

```

字段说明同 non-config，配置详情请参考插件文档。

### 3. 验证安装

通过标签在 global cluster 中查找 ModuleInfo，查看状态和版本：

```

# 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

字段说明同 non-config。

## 升级流程

升级已安装插件到新版本：

#### 1. 上传新版本：

- 按照相同步骤将新版本上传至平台。

## 2. 验证新版本：

- 进入 管理员 > **Marketplace** > **Upload Packages**
- 切换到 **Cluster Plugin** 标签页
- 插件详情会显示新上传的版本

## 3. 执行升级：

- 进入 管理员 > **Clusters** > **Clusters**
- 可升级插件的集群会显示升级图标
- 进入集群详情，切换到 **Features** 标签页
- 功能组件下的升级按钮会被激活
- 点击 **Upgrade** 完成插件升级

# Upload Packages

平台提供了一个命令行工具 `violet`，用于将从 Custom Portal 的 Marketplace 下载的软件包上传到平台。

`violet` 支持上传以下类型的软件包：

- **Operator**
- 集群插件
- **Helm Chart**

当 **Cluster Plugins** 或 **OperatorHub** 中的软件包状态显示为 `Absent` 时，您需要使用该工具上传对应的软件包。

`violet` 的上传流程主要包括以下步骤：

1. 解包并提取软件包信息
2. 将镜像推送到镜像仓库
3. 在平台上创建 **Artifact** 和 **ArtifactVersion** 资源

---

## 目录

下载工具

Linux 或 macOS

Windows

前置条件

使用方法

`violet show`

`violet push`

---

上传 Operator 到多个集群

上传 Operator 到备用 global 集群

上传集群插件

上传 Helm Chart 到 chart 仓库

仅推送目录中所有软件包的镜像

仅创建目录中所有软件包的 CR 资源

---

## 下载工具

登录 **Custom Portal**，进入 **Downloads** 页面，点击 **CLI Tools**。下载与您的操作系统和架构匹配的二进制文件。

下载完成后，将工具安装到您的服务器或 PC 上。

## Linux 或 macOS

非 **root** 用户：

```
# 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
```

**root** 用户：

```
# 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
```

## Windows

1. 下载文件并重命名为 `violet.exe`，或者使用 PowerShell 重命名：

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

2. 在 PowerShell 中运行该工具。

注意：如果工具路径未添加到环境变量，运行命令时必须指定完整路径。

## 前置条件

### 权限要求

- 必须提供有效的平台用户账号（用户名和密码）。
- 账号的 `role` 属性必须设置为 `System`，且角色名称必须为 `platform-admin-system`。

注意：如果账号的 `role` 属性设置为 `Custom`，则无法使用该工具。

## 使用方法

### `violet show`

上传软件包前，可使用 `violet show` 命令预览软件包详情。

```
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
...]
```

## violet push

以下示例展示了常见的使用场景。

在深入示例前，以下是命令中常用的一些可选参数：

```

--platform-address <platform access URL> # 平台访问地址, 例如 "https://example.com"
--platform-username <platform user> # 平台用户的用户名
--platform-password <platform password> # 平台用户的密码
--clusters <cluster names> # 指定目标集群, 多个集群用逗号分隔 (例如
region1,region2)

--dest-repo <image repository URL> # 指定目标镜像仓库地址。上传扩展到备用集群时
必须指定。

# 当指定 `--dest-repo` 时, 必须提供镜像仓
库的认证信息或使用 `--no-auth`。
--username <registry user> # 指定镜像仓库的用户名。
--password <registry password> # 指定镜像仓库的密码。
--no-auth # 指定镜像仓库无需认证。
--plain # 指定镜像仓库使用 HTTP 而非 HTTPS。

--skip-crs # 跳过创建 `Artifact` 和 `ArtifactVersion`
资源, 仅推送镜像。

# 这样可以避免 Operator 或集群插件在 <Term
name="productShort" /> 升级过程中被提前更新。
--skip-push # 仅创建 `Artifact` 和 `ArtifactVersion` 资
源, 不推送镜像。

```

## 上传 Operator 到多个集群

```

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

- 如果未指定 `--clusters`, Operator 默认上传到 **global** 集群。 \

## 上传 Operator 到备用 global 集群

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

## 上传集群插件

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

### INFO

- 上传集群插件时无需指定 `--clusters` 参数，平台会根据亲和性配置自动分发。如果指定了 `--clusters`，该参数将被忽略。

## 上传 Helm Chart 到 chart 仓库

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

### INFO

- Helm Chart 只能上传到平台提供的默认 `public-charts` 仓库。

## 仅推送目录中所有软件包的镜像

当从 Marketplace 下载多个软件包时，可以将它们放在同一目录下，一次性上传：

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

## 仅创建目录中所有软件包的 **CR** 资源

当从 Marketplace 下载多个软件包时，可以将它们放在同一目录下，一次性上传：

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