# 扩展

概览 | **Operator** | 集群插件

图表仓库 | 下载软件包 | 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，使用户能够轻松地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，为平台提供专门的运维能力。这些扩展使得在集群内自动化管理复杂应用和服务成为可能。
- 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。该系统相比标准方法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，以满足特定的运维需求和使用场景。

# Operator

## 目录

---

# 概览

基于 **OLM (Operator Lifecycle Manager)** 框架，**OperatorHub** 提供了用于管理 Operators 安装、升级和生命周期的统一界面。
管理员可以使用 OperatorHub 安装和管理 Operators，从而为 Kubernetes 应用实现完整的生命周期自动化，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 catalog，并生成相应的 InstallPlan。
- **CatalogSource**：一种命名空间级别的 CRD，用于管理 Operator catalog 来源并提供 Operator 元数据（例如版本信息、受管 CRD）。平台提供 3 个默认的 CatalogSource：**system**、**platform** 和 **custom**。**system** 中的 Operators 不会显示在 OperatorHub 中。
- **ClusterServiceVersion (CSV)**：一种命名空间级别的 CRD，用于描述 Operator 的特定版本，包括它所需的资源、CRD 和权限。
- **Subscription**：一种命名空间级别的 CRD，用于描述所订阅的 Operator、其来源、获取通道和升级策略。
- **InstallPlan**：一种命名空间级别的 CRD，用于描述需要执行的实际安装操作（例如创建 Deployments、CRD 和 RBAC）。只有在 InstallPlan 获得批准后，Operator 才会被安装或升级。

# **Operator** 来源

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

1. 灵雀云 由 灵雀云 提供和维护，包括完整的生命周期管理、安全更新、技术支持和 SLA 承诺。
2. **Curated** 从开源社区中精选，保持与社区版本一致，不修改代码，也不重新编译。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，定期更新以确保可安装性，但不保证功能完整性；不提供 SLA 或 灵雀云 支持。
4. **Marketplace** 由经过 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，而核心维护由厂商负责。
5. **Custom** 由用户自行开发并上传，以满足自定义使用场景需求。

# 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：

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

推荐使用 **Cluster** 模式（**AllNamespaces**）。 平台最终将升级到 OLM v1，而 OLM v1 仅支持 AllNamespaces 安装模式。因此，应强烈避免使用 SingleNamespace 和 MultiNamespace。

## 更新通道

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

## 批准策略

可选项：**Automatic** 或 **Manual**。

- **Automatic**：当所选通道中发布新版本时，OLM 会自动升级 Operator。
- **Manual**：当有新版本可用时，OLM 会创建一个升级请求，必须由集群管理员手动批准后才会执行升级。

注意：来自 灵雀云 的 Operators 仅支持 **Manual** 模式；否则安装将失败。

### 安装位置

建议为每个 Operator 创建一个单独的命名空间。

如果多个 Operators 共享同一个命名空间，它们的 Subscriptions 可能会被解析到同一个 InstallPlan 中：

- 如果该命名空间中的某个 InstallPlan 需要 Manual 批准且仍处于 pending 状态，它可能会阻塞同一 InstallPlan 中其他 Subscriptions 的自动升级。

## 通过 Web Console 安装

1. 登录 Web Console，并切换到 **Administrator** 视图。
2. 导航到 **Marketplace > OperatorHub**。
3. 如果状态为 **Absent**：
   - 从 灵雀云 Customer Portal 下载 Operator package，或联系支持团队。
   - 使用 `violet` 将 package 上传到目标集群（参见 CLI）。
   - 在 **Marketplace > Upload Packages** 页面，切换到 **Operator** 选项卡并确认上传。
4. 如果状态为 **Ready**，单击 **Install**，然后按照 Operator 的用户指南操作。

## 通过 YAML 安装

以下示例演示了来自 灵雀云（仅支持 Manual）和非 灵雀云 来源（支持 Manual 或 Automatic）的 Operator 安装方式。

**INFO**

与 cluster 插件不同（使用 YAML 时必须始终安装在 **global cluster** 中），Operator 会安装到你希望其运行的 目标集群 中。在执行任何 YAML manifest 之前，请确保你已连接到目标集群。

# 手动

`harbor-ce-operator` 来自 灵雀云，且仅支持 **Manual** 批准。
在 Manual 模式下，即使发布了新版本，Operator 也不会自动升级。你必须先手动 **Approve**，然后 OLM 才会执行升级。

## 1. 检查可用版本

```
(
  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
```

这表示 `harbor-ce-operator` 来自 `platform` catalogSource。

### 3. 创建命名空间

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

### 4. 创建 Subscription

```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
```

字段说明：

- **annotation** `cpaas.io/target-namespaces`：建议设置为空；为空表示集群范围安装。
- **.metadata.name**：Subscription 名称（符合 DNS 规范，最长 253 个字符）。
- **.metadata.namespace**：Operator 将要安装到的命名空间。
- **.spec.channel**：订阅的 Operator 通道。

- **.spec.installPlanApproval**：批准策略（`Manual` 或 `Automatic`）。这里 `Manual` 表示安装/升级需要手动批准。
- **.spec.source**：Operator catalogSource。
- **.spec.sourceNamespace**：必须设置为 cpaas-system，因为平台提供的所有 catalogSource 都位于该命名空间中。
- **.spec.startingCSV**：指定 Manual 批准时要安装的版本；如果为空，则默认使用该通道中的最新版本。Automatic 模式下不需要此字段。

## 5. 检查 Subscription 状态

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

关键信息输出：

- **.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；必须先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-s
ubs -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="$(kubectl -n harbor-ce-operator get subscription harbor-ce-operator
-subs -o jsonpath='{.status.installPlanRef.name}')"
kubectl -n harbor-ce-operator patch installplan "$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
PHASE
harbor-ce-operator.v2.12.11   Alauda Build of Harbor   2.12.11   harbor-c
e-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

`clickhouse-operator` 来自非 灵雀云 来源，其批准策略可以设置为 **Automatic**。
在 Automatic 模式下，当发布新版本时，Operator 会自动升级，无需手动批准。

### 1. 检查可用版本

```
(
  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：

```
platform
```

这表示 `clickhouse-operator` 来自 `platform` catalogSource。

## 3. 创建命名空间

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

## 4. 创建 Subscription

```yaml
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
```

字段说明与 Manual 中相同。

### 5. 检查 Subscription 状态

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

## 高级 Subscription 配置

以下配置适用于批准策略为 `Manual` 或 `Automatic` 的 Subscriptions。

# 配置 **Operator Pod** 调度

要控制 Operator Pod 运行的位置，可在 `Subscription.spec.config` 中配置调度设置。
该配置仅影响 OLM 管理的 Operator deployment。
它不控制 Operator 安装后创建的实例或自定义资源的调度。
要控制这些工作负载，请参考对应 Operator 的文档。

建议：当你希望 Operator 运行在 infra 节点或其他专用节点上时，使用此配置。

前提条件：目标节点必须已具备所需标签。如果目标节点带有 taint，请配置相应的 toleration。

可用字段：

- **.spec.config.nodeSelector**：可选。选择 Operator Pod 可以运行的节点。
- **.spec.config.tolerations**：可选。当 taint 匹配时，允许 Operator Pod 运行在带 taint 的节点上。

`Subscription` 示例片段：

```
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
      - key: node-role.kubernetes.io/infra
        operator: Equal
        value: reserved
        effect: NoSchedule
```

如果 Operator 已经安装，可以 patch 现有的 `Subscription`：

```
kubectl patch subscription <subscription-name> -n <operator-namespace> --
type='merge' -p '
{
  "spec": {
    "config": {
      "nodeSelector": {
        "node-role.kubernetes.io/infra": ""
      },
      "tolerations": [
        {
          "effect": "NoSchedule",
          "key": "node-role.kubernetes.io/infra",
          "operator": "Equal",
          "value": "reserved"
        }
      ]
    }
  }
}'
```

# 升级流程

升级流程从上传新版本 **Operator** 开始。

上传完成后，请等待大约 **10–15** 分钟，以便平台同步新版本信息。

同步完成后，升级将按照 **Subscription** 中配置的策略执行：

- 如果 Operator 的 **Approval Strategy** 设置为 **Automatic**，则 Operator 会自动升级。
- 如果策略设置为 **Manual**，则必须手动批准升级请求。你可以选择以下升级方式之一：
  - **Batch Upgrade**：在 **Platform Management > Cluster Management > Cluster > Features** 页面执行升级。
  - **Individual Upgrade**：在 **OperatorHub** 中手动批准升级请求。

注意：只有 灵雀云 提供的 Operators 支持批量升级。

# Cluster Plugin

## 目录



## 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，提供插件的可定制配置选项或参数组合。例如，安装 灵雀云容器平台 Log Collector 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，则该字段为空。

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。
- 同一集群中，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：**Administrator** > **Marketplace** > **Cluster Plugins**

此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 灵雀云 Customer Portal 下载对应插件包。
   - 若无访问权限，请联系技术支持。

2. 上传包到平台：

   - 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 **Administrator** > **Marketplace** > **Upload Packages**
   - 切换到 **Cluster Plugin** 标签页
   - 找到已上传的插件名称
   - 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 Install 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。
>
> 虽然插件本身可根据 ModuleConfig 中的亲和性设置，目标为 global cluster 或 workload cluster，但 `ModuleInfo` 资源只能在 **global cluster** 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`，且已发布版本 `v4.0.4`。

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，表示 `web-cli` 只能安装在 global cluster。`.spec.config` 为空，说明插件无需配置，可直接安装。

## 2. 创建 ModuleInfo

在 global cluster 创建 ModuleInfo 资源，安装插件且无配置参数：

```yaml
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 的亲和性冲突，安装会失败。
  注意： 此标签不改变 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`，缺失此字段会导致安装失败。
- `.spec.config`：对应 ModuleConfig 为空时，此字段可留空。
- `.spec.version`：指定安装的插件版本，必须与 ModuleConfig 中的 `.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 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_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
```

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`，且已发布版本 `v4.0.15`。

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：

- 该插件仅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` 时才会安装对应驱动。

## 2. 创建 ModuleInfo

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

```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
```

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 查找 ModuleInfo，查看状态和版本：

```
# 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
```

字段说明同 non-config。

## 集群插件生命周期值

**Cluster Plugins** 页面使用 `cpaas.io/lifecycle-type` 标签指示集群插件的维护和升级方式。若标签缺失，则插件视为 `Agnostic`。

其中，`Core` 表示插件随 ACP Core 版本及集群 Distribution Version 一起发布和升级。

**Cluster Plugins** 列表页显示 **Life cycle** 列，并提供 **Life cycle** 过滤器，方便快速识别各插件类型。

| 生命周期 | 描述 | 升级路径 |
| --- | --- | --- |
| `Core` | 插件随 ACP Core 版本及集群 Distribution Version 一起发布和升级，不支持单独升级。 | 升级集群以更新插件。 |
| `Aligned` | 插件随 ACP 发布流发布，但可在发布新版本时独立升级。 | 在 Cluster Plugins 列表页或详情页进行插件升级。 |
| `Agnostic` | 插件独立于 ACP 发布。 | 在 Cluster Plugins 列表页或详情页进行插件升级。 |

# 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。
   - 上传完成后，等待约 **10–15** 分钟，平台同步新版本信息。
2. 验证新版本：
   - 进入 **Administrator** > **Marketplace** > **Upload Packages**
   - 切换到 **Cluster Plugin** 标签页
   - 插件详情显示新上传版本
3. 查看插件：
   - 进入 **Administrator** > **Marketplace** > **Cluster Plugins**
   - 选择目标集群
   - 在列表页或插件详情页查看插件信息
4. 执行升级：
   - 若插件生命周期为 `Aligned` 或 `Agnostic`，可在列表页或详情页启动单独升级

- 平台将已安装插件升级到新发布版本

5. 处理 `Core` 插件：

   - `Core` 插件不支持单独升级
   - 检测到新版本时，插件详情页会提示升级集群

注意： `Upgrade` 指更改已安装插件版本， `Update` 仍为插件特定操作，用于修改配置或执行插件定义的更新逻辑。

# Chart Repository

有关 Chart 仓库和 Helm charts 的信息，请参见 Working with Helm Charts。

# 下载软件包

该平台提供了一个命令行工具 `violet`，用于从平台下载软件包。

## 目录

violet ac import-yaml

可选标志

示例工作流

---

# 下载工具

登录到 灵雀云 **customer portal**，进入 **Downloads** 页面，然后单击 **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，将该工具安装到服务器或 PC 上。

## 适用于 **Linux** 或 **macOS**

对于非 **root** 用户：

```
# 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
```

对于 **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
```

## 适用于 **Windows**

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

   ```
   # 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中，则在运行命令时必须指定完整路径。

对于本页所述的工作流，请使用 **Violet v4.2.13** 或更高版本。安装二进制文件后，请验证其版本：

```
violet version
```

# 前提条件

权限要求

- 您必须提供有效的平台用户账户（account、username 和 password）。

# 用法

# violet ac login

在下载软件包之前，请先使用 `violet ac login` 命令登录平台。

```
violet ac login --account=<account> --username=<username> --password=<password> --ac-url=<url>
# or provide an existing token:
violet ac login --access-token=<token> --ac-url=<url>
```

## 可选标志

```
--account         account / tenant name
--username        username
--password        password
--ac-url          AC system URL (default: `https://cloud.alauda.io`)
--access-token    directly provide an access token to skip username/password login
```

> **Note**
>
> 您可以从 `Customer Portal - Settings` 导出 access token。 access token 的有效期为登录成功后 24 小时内。

# violet ac scenarios

列出可用的 scenarios
输出：一个格式化表格，列出 `ID`、`Name` 和 `Description`。

```
violet ac scenarios --arch=amd64 --platformVersion=v4.4.0
```

## 可选标志

```
--arch                target architecture (`amd64` , `arm64` , `hybrid`, d
efault: `amd64`)
--platformVersion     target platform version
```

## ACP 4.4 安装和升级场景

在准备 ACP 4.4 软件包时，请使用以下 scenarios：

| 场景 | 用途 |
| --- | --- |
| `Common ACP Extensions` | 下载包含新 ACP 4.4 安装所需的八个 Aligned applications 以及其他 Extensions 的软件包集合。 |
| `ACP Upgrade to v4.4` | 在将现有环境升级到 ACP 4.4 时，下载所需的 Aligned Extension 软件包。 |

将 `<target-platform-version>` 设置为 Core Package 的确切 ACP Distribution Version，例如 `v4.4.0`。将 `<arch>` 设置为 `amd64`、`arm64` 或 `hybrid`，以匹配 Core Package。

请先列出可用 scenarios，以确认目标版本和架构已发布预期的 scenario。然后下载所需的 scenario：

```
violet ac scenarios --arch=<arch> --platformVersion=<target-platform-vers
ion>

# New installation
violet ac download-app --arch=<arch> --platformVersion=<target-platform-v
ersion> --scenario="Common ACP Extensions"

# Upgrade to ACP 4.4
violet ac download-app --arch=<arch> --platformVersion=<target-platform-v
ersion> --scenario="ACP Upgrade to v4.4"
```

下载 scenario 不会将其软件包添加到 Core Package 中。请解压 Core Package，并将安装或升级操作步骤所需的软件包手动复制到其 `plugins/` 目录中。有关确切的应用名称和复制步骤，请参见 Download 和 Pre-Upgrade Preparation。

# violet ac packages

列出可用软件包。
输出：一个格式化表格，列出 `APP ID`、`APP Name`、`Channel And Version` 和 `Package`。

```
# download all packages for the specified architecture, platform version, and scenario
violet ac packages --arch=<arch> --platformVersion=<version> --scenario=<scenario>
# download all packages for the specified architecture and platform version
violet ac packages --arch=<arch> --platformVersion=<version>
# download single package for a specific application ID
violet ac packages --appID=my-app
```

## 可选标志

```
--arch               target architecture (`amd64` , `arm64` , `hybrid`, default: `amd64`)
--platformVersion    target platform version
--appID              download single app for a specific application ID
--scenario           scenario filter (optional)
```

> **Note**
>
> 关于 `scenario`，如果未配置，则会显示所有软件包。

# violet ac download-pkg

下载特定架构和平台版本的软件包及其签名文件。

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

## 可选标志

```
--arch              target architecture (`x86` , `arm` , `hybrid`, defau
lt: `x86`)
--platformVersion   target platform version
--type              package type (`core`, `extensions`, `standard`, defa
ult: `core`)
```

> 注意：关于 `type`，对于 v4.0.5 及更高版本，工具默认下载 core package — 您无需设置此选项。
> 对于 v4.0.0 到 v4.0.4，默认值为 `core`，用于下载 core package；您可以将该选项设置为 `extensions` 以下载 extension packages。

## violet ac download-app

按 scenario（批量）或通过指定 `appID` 和 `appVersion` 下载 application packages。该命令会获取下载 URL，然后下载 package 和 checksum 文件。

```
violet ac download-app --arch=amd64 --platformVersion=<version> --scenari
o=<scenario>
# List available packages status
violet ac download-app --arch=amd64 --platformVersion=<version> --scenari
o=<scenario> --check=true
# or download a specific app version
violet ac download-app --arch=amd64 --appID=<app-id> --appVersion=<app-ve
rsion1>,<app-version2>
```

### 可选标志

```
--arch              target architecture (`amd64` , `arm64` , `hybrid`, d
efault: `amd64`)
--platformVersion   target platform version
--appID             specific application ID
--appVersion        application version (supports multiple versions sepa
rated by commas)
--check             If true, do not download; only check available packa
ges status (default: `false`)
--scenario          scenario name (optional, download latest versions fo
r the scenario)
```

**Note**

关于 `scenario`，如果未配置，则会下载所有软件包。

## violet ac download-os

从 AC Marketplace OS 页面下载 OS packages。请先使用 `--list` 查看可用的平台版本、架构和操作系统，然后下载匹配的 OS package 和 checksum 文件。

```
# List all available OS package filters
violet ac download-os --list
# List OS packages for a specific platform version
violet ac download-os --platformVersion=v4.1.0 --list
# Download matching OS packages
violet ac download-os --platformVersion=v4.1.0 --arch=x86_64 --operatingS
ystem=ubuntu
```

### 标志

```
--platformVersion    target platform version (required when downloading)
--arch               target architecture (required when downloading)
--operatingSystem    target operating system (required when downloading)
--list               list matching OS packages without downloading (defau
lt: `false`)
```

**Note**

在下载之前，请使用带更少筛选条件的 `--list` 来发现有效的 `platformVersion`、`arch` 和 `operatingSystem` 值。

## violet ac import-yaml

读取一个本地 YAML 文件（默认 `./apps.yaml`，`violet list` 导出），其中包含一个 `applications` map，将其发送到 AC scenario-check endpoint，并显示验证结果。也可选择

下载已验证的软件包。

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

### 可选标志

```
--arch                target architecture (`amd64` , `arm64` , `hybrid`, d
efault: `amd64`)
--platformVersion     target platform version
--download            boolean; if true, attempt to download validated pack
ages after the check
--path                path to the YAML file (default `./apps.yaml`)
```

## 示例工作流

- 登录并保存 token：

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

- 列出可用 scenarios：

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

- 下载目标版本的 core packages：

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

- 列出可用软件包并下载某个 scenario 的最新版本：

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

- 列出可用软件包状态（`Downloaded` 或 `Download Failed`）：

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

- 列出并下载 OS packages：

```
violet ac download-os --platformVersion=v4.1.0 --list
violet ac download-os --platformVersion=v4.1.0 --arch=x86_64 --operatin
gSystem=ubuntu
```

- 验证 applications YAML 并下载已验证的软件包：

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

# 上架软件包

平台提供了一个命令行工具 `violet`，用于将从 灵雀云 Customer Portal 中下载的 Marketplace 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** 中某个软件包的状态显示为 `Absent` 时，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. 从软件包中提取并获取信息
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

登录 灵雀云 **Customer Portal**，进入 **Downloads** 页面，然后单击 **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 /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
```

**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. 下载文件并将其重命名为 `violet.exe`，或者使用 PowerShell 进行重命名：

   ```
   # 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中，则运行命令时必须指定完整路径。

请使用 **Violet v4.2.13** 或更高版本 运行本页所述的工作流。安装二进制文件后，请验证其版本：

```
violet version
```

# 前提条件

权限要求

- 您必须提供有效的平台凭据，可以是用户名和密码，也可以是平台令牌。
- 该账户的 role 属性必须设置为 `System`，且 role 名称必须为 `platform-admin-system`。

> **注意：**如果您的账户的 role 属性设置为 `Custom`，则无法使用该工具。

# 用法

## 通用参数

多个 `violet` 命令接受以下参数。具体用法请参见各命令章节。

### 平台连接参数

```
--platform-address <platform access URL>     # The access URL of the plat
form, e.g., "https://example.com"
--platform-username <platform user>          # The username of the platfo
rm user
--platform-password <platform password>      # The password of the platfo
rm user
--platform-token <platform token>            # The platform access token;
cannot be used together with username/password
```

要创建或管理平台令牌，请登录平台 Web Console，打开 **Profile** > **API Tokens**，并创建一个具有合适过期时间的令牌。详细信息请参见 API Authentication。

> **WARNING**

身份提供方（**IdP**）用户必须使用平台令牌。 用户名和密码登录仅会对平台本地用户存储进行身份验证。如果您的账户来自外部 Identity Provider（例如 LDAP 或 OIDC），使用用户名和密码登录会失败——请改用 `--platform-token`。

**Violet 版本早于 v4.2.13**

早期版本在使用 `--platform-token` 时可能会失败，并报错 `failed to get user info from platform server`。在使用本页所述的基于令牌的工作流之前，请先升级到 Violet v4.2.13 或更高版本。

## 镜像仓库参数

```
--dest-repo <image repository address>       # Specify the destination im
age repository address, e.g., "harbor.demo.io"
--username <registry user>                   # The username of the specif
ied image registry
--password <registry password>               # The password of the specif
ied image registry
--no-auth                                    # Specify if the image regis
try does not require authentication
--plain                                      # Specify if the image regis
try uses HTTP instead of HTTPS
```

**WARNING**

**IPv6** 地址限制

对于 `--platform-address` 和 `--dest-repo` 参数：

- 如果使用的是 IP 地址（而不是域名），则不支持 **IPv6** 格式
- 仅支持 IPv4 地址或域名

# violet show

在上传软件包之前，请使用 `violet show` 命令预览其详细信息。

```
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

在升级平台时，使用 `violet list` 列出已安装的扩展，并将结果导出到 YAML 文件。随后可以将生成的文件上传到 Alauda Cloud，以便下载所需的扩展软件包。

`violet list` 覆盖生命周期为 **Aligned** 或 **Agnostic** 的扩展。**Core** 插件会随着集群升级，不通过 `violet list` 管理。

该命令会列出以下应用类型：

- **ModulePlugin** — 生命周期类型为 `aligned` 或 `agnostic` 的集群插件。
- **OperatorBundle** — 已安装的 Operator。
- **Chart** — 根据 chart 引用创建的 Helm Chart 应用。

默认情况下，该命令会将 YAML 内容打印到 stdout，并将其写入当前目录中的 `apps.yaml`。

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

您也可以使用平台令牌进行认证，而不是使用用户名和密码：

```
violet list \
  --platform-address "https://example.com" \
  --platform-token "<platform_token>"
```

如需仅导出安装在特定集群中的应用，请使用逗号分隔多个集群名称：

```
violet list \
  --platform-address "https://example.com" \
  --platform-token "<platform_token>" \
  --clusters "global,region1" \
  --output-file "./apps.yaml"
```

示例输出：

```
applications:
  topolvm-operator:
    displayName: NativeStor
    type: OperatorBundle
    installed:
      - clusterName: global
        version: 2.3.0
  log-center:
    displayName: Log Center
    type: ModulePlugin
    installed:
      - clusterName: region1
        version: v1.2.0
```

## 可选标志

```
--clusters <cluster names>                    # List applications installe
d in the specified clusters, separated by commas
--output-file <output file>                   # File path for the output a
pplication list (default: "apps.yaml")
--debug                                       # Use debug log level
```

关于平台连接参数（`--platform-address`、`--platform-username`、`--platform-password`、`--platform-token`），请参见 通用参数。

# violet gc

使用 `violet gc` 清理未安装的生命周期为 **Aligned** 或 **Agnostic** 的 Operator 和集群插件资源。该命令会删除不再安装或不再与已安装版本匹配的资源。

该命令可以删除以下资源类型：

- **Artifact** — 未安装的 operator 的 artifact。
- **ArtifactVersion** — 已安装 operator 的旧 artifact 版本。会保留当前已安装的 CSV 或版本。
- **ModulePlugin** — 未安装在任何集群中的集群插件。
- **ModuleConfig** — 过时的集群插件配置记录。

`violet gc` 通过检查 global 集群中的 `OperatorView` 和 `ModulePluginView` 记录来确定清理对象。

> **DANGER**
>
> `violet gc` 会从 global 集群中删除资源，不提供 dry-run 模式或撤销操作。请仅在文档化的维护窗口期间运行，并在使用 `--clusters` 前确认目标集群。

默认情况下，该命令会扫描平台用户可见的所有集群，并打印已删除资源的摘要。

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

您也可以使用平台令牌进行认证，而不是使用用户名和密码：

```
violet gc \
  --platform-address "https://example.com" \
  --platform-token "<platform_token>"
```

如需仅清理与特定集群相关的资源，请使用逗号分隔多个集群名称：

```
violet gc \
  --platform-address "https://example.com" \
  --platform-token "<platform_token>" \
  --clusters "global,region1"
```

示例输出：

```
GC Summary:
TYPE                 NAME                                                C
LUSTER
-------------------------------------------------------------------------
-----------------
Artifact             opensearch-operator                                 r
egion1
ArtifactVersion      opensearch-operator.v2.8.0                          r
egion1
ModuleConfig         log-center-default                                  g
lobal
```

如果未删除任何资源，命令会输出：

```
No resources were deleted.
```

### 可选标志

```
--clusters <cluster names>                    # Limit cleanup to the speci
fied clusters, separated by commas
--debug                                       # Use debug log level
```

关于平台连接参数（`--platform-address`、`--platform-username`、`--platform-password`、`--platform-token`），请参见 通用参数。

## violet verify

使用 `violet verify` 命令在上传前验证一个或多个软件包的签名。 支持两种验证方式：**checksum** 和 **GPG**。 软件包（`.tgz`）及其对应的签名文件必须位于同一目录中。

```
violet verify example.tgz
# or verify all packages within a directory
violet verify packages_dir_name
```

示例输出：

```
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
```

说明：

- **Verified successfully with GPG** — 列出的文件已成功使用 **GPG** 签名文件（`.sig` 扩展名）完成验证。
- **Verified successfully with checksum** — 使用 checksum 文件（例如 `.sha256`）验证的文件通过了完整性检查。
- **Verification failed** — 列出的文件因签名不匹配或无效而验证失败。
- **No verification file found** — 目录中未找到对应的 `.sig`（GPG）或 checksum 文件。

### 可选标志

```
--debug       Use debug log level.
-h, --help    Display help information for the verify command.
```

## violet push

以下示例展示了常见使用场景。

关于平台连接和镜像仓库参数，请参见 通用参数。

对于非交互式使用，建议优先使用 `--platform-token` 而不是 `--platform-password`。请参见 通用参数。

## 可选标志

```
--clusters <cluster names>                    # Specify target clusters, s
eparated by commas (e.g., region1,region2)
--target-catalog-source <catalog source>      # Target CatalogSource for o
perators: "platform", "system", or a custom CatalogSource name (default:
"platform")
--target-chartrepo <chart repository>         # Target chart repository fo
r Helm charts (default: "public-charts", the platform-managed Helm reposi
tory)
--skip-crs                                    # Skip creating custom resou
rces in the cluster
--skip-push                                   # Skip pushing images and ar
tifacts to the registry
```

`--skip-crs` 和 `--skip-push` 不能同时指定。

当指定 `--dest-repo` 时，必须提供镜像仓库的认证信息或 `--no-auth`。

## 将 **Operator** 上传到多个集群

```
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`，则默认将 Operator 上传到 **global** 集群。\

## 将 Operator 上传到备用 global 集群

```
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**

使用 `violet` 将软件包上传到备用集群时：

- 必须指定参数 `--dest-repo <备用集群的 VIP 地址>`
- 必须将参数 `--platform-address` 设置为 备用集群 的平台访问地址
- 必须提供备用集群镜像仓库的认证信息或 `--no-auth` 参数

否则，软件包将上传到 主集群 的镜像仓库，从而导致备用集群无法安装或升级扩展。

## 上传集群插件

```
violet push plugins-cloudedge-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，因为平台会根据其亲和性配置自动分发。 如果指定了 `--clusters` ，该参数将被忽略。\

## 将 Helm Chart 上传到 chart 仓库

```
violet push plugins-cloudedge-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。\

## 一次推送所有软件包

当从 Marketplace 下载了多个软件包时，可以将它们放在同一目录中并一次全部上传：

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

**WARNING**

当升级目标是 **global** 集群 时，可以省略 `--clusters` 参数，因为其默认值就是上传到 global 集群。

但是，当升级目标是业务集群时，您必须指定 `--clusters <业务集群名称>` 参数。