

ACP CLI (ac)

[Getting Started with ACP CLI](#)

[配置 ACP CLI](#)

[ac 和 kubectl 命令的使用](#)

[管理 CLI 配置文件](#)

[使用插件扩展 ACP CLI](#)

[AC CLI 开发者命令参考](#)

[AC CLI 管理员命令参考](#)

Getting Started with ACP CLI

目录

About ACP CLI

Installation

Installing from Binary

Installing ACP CLI on Linux

Installing ACP CLI on macOS

Installing ACP CLI on Windows

First Steps

Logging into ACP Platform

Interactive Login

Login Using Parameters

Login Using Environment Variables

Quick Configuration Management

View Current Status

Switch Clusters

Switch Namespaces

Basic Resource Operations

Managing Multiple Environments

Your First Application

Create a Simple Pod

View Application Status

Clean Up

Getting Help

Built-in Help System

General Help

Command-Specific Help

Resource Documentation

Logging Out

About ACP CLI

通过 ACP CLI (`ac`)，您可以从终端管理 ACP 平台和集群。ACP CLI 提供类似 `kubectl` 的体验，针对 ACP 的集中式、基于代理的多集群架构进行了优化。

ACP CLI 适用于以下场景：

- 从统一界面管理 ACP 平台和多个集群
- 直接操作项目源代码，脚本化 ACP 平台操作并自动化 workflows
- 在带宽受限且无法使用 Web 控制台时管理项目
- 跨不同 ACP 环境（生产、预发布、开发）管理应用

Installation

Installing from Binary

您可以通过下载适用于您操作系统的二进制文件来安装 ACP CLI (`ac`)。

请按照以下步骤下载正确的软件包：

1. 在浏览器中打开 [Alauda Cloud download page](#) ↗。
2. 选择 **CLI Tools** 进入 CLI 下载页面。
3. 找到 **ACP CLI (ac)** 部分。
4. 下载与您的操作系统和 CPU 架构匹配的二进制文件（例如，`ac-linux-amd64`）。

Installing ACP CLI on Linux

1. 按上述步骤完成下载 Linux 版本二进制文件（例如，`ac-linux-amd64` 或 `ac-linux-arm64`）。
2. 赋予二进制文件可执行权限：

```
chmod +x ac-linux-amd64
```

将 `ac-linux-amd64` 替换为您下载的文件名。

3. 将二进制文件移动到 PATH 中并重命名为 `ac`：

```
sudo mv ac-linux-amd64 /usr/local/bin/ac
```

如果下载了其他版本，请相应调整文件名。

4. 验证安装：

```
ac version
```

Installing ACP CLI on macOS

1. 按上述步骤完成下载 macOS 版本二进制文件（例如，`ac-darwin-amd64` 或 `ac-darwin-arm64`）。
2. 赋予二进制文件可执行权限：

```
chmod +x ac-darwin-amd64
```

将 `ac-darwin-amd64` 替换为您下载的文件名。

3. 将二进制文件移动到 PATH 中并重命名为 `ac`：

```
sudo mv ac-darwin-amd64 /usr/local/bin/ac
```

如果下载了其他版本，请相应调整文件名。

4. 验证安装：

```
ac version
```

Installing ACP CLI on Windows

1. 按上述步骤完成下载 Windows 版本二进制文件（例如， `ac-windows-amd64.exe` ）。
2. 将 `ac-windows-amd64.exe` 移动到 PATH 中的某个目录，并根据需要重命名为 `ac.exe` 。如果您希望保留原名，也可以，只需确保该文件所在目录在 PATH 中。
3. 验证安装：

```
ac version
```

First Steps

Logging into ACP Platform

`ac login` 命令是连接 ACP 平台的入口。它负责身份验证并自动配置对所有可用集群的访问。

Interactive Login

为了获得最简单的体验，直接运行 `ac login` 并按照交互提示操作：

```
$ ac login
Platform URL: https://prod.acp.com
Session name: prod
Username: user@example.com
Password: [hidden]
✓ Login successful. Welcome, user@example.com!

Your kubeconfig has been configured for the 'prod' platform.
+ Default context 'prod/global' has been created and activated.

To switch clusters within this session, use:
  ac config use-cluster <cluster_name>

To switch between platforms, use:
  ac config get-sessions          # Discover all configured sessions
  ac config use-session <name>   # Switch to different platform
```

Login Using Parameters

您也可以直接通过参数登录：

```
ac login https://prod.acp.com --name prod --username user@example.com
```

Login Using Environment Variables

用于自动化和脚本时，可以使用环境变量：

```
export AC_LOGIN_PLATFORM_URL=https://prod.acp.com
export AC_LOGIN_SESSION=prod
export AC_LOGIN_USERNAME=user@example.com
export AC_LOGIN_PASSWORD=your-password
ac login
```

Quick Configuration Management

登录后，ACP CLI 提供便捷命令以支持日常操作：

View Current Status

使用 `ac namespace` 查看当前操作上下文：

```
$ ac namespace
You are currently in namespace "default" (no namespace set in context).

Context:  prod/global
Cluster:  acp:prod:global
Server:   https://acp.prod.example.com/kubernetes/global/
```

Switch Clusters

在当前会话中切换集群：

```
$ ac config use-cluster workload-a
Switched to context "prod/workload-a".

$ ac config use-cluster global
Switched to context "prod/global".
```

Switch Namespaces

更改当前命名空间：

```
$ ac namespace my-app-dev
Now using namespace "my-app-dev" in context "prod/global".
```

Basic Resource Operations

使用标准 kubectl 命令管理资源：

```
# 列出当前命名空间中的 pods
$ ac get pods

# 查看指定 pod 详情
$ ac describe pod my-pod

# 获取所有命名空间的服务
$ ac get services --all-namespaces

# 应用配置文件
$ ac apply -f deployment.yaml
```

Managing Multiple Environments

对于管理多个 ACP 平台的用户：

列出所有已配置的会话：

```
$ ac config get-sessions
```

CURRENT	SESSION	PLATFORM	USER	CLUSTERS
*	prod	https://acp.prod.example.com	user@example.com	3
	staging	https://staging.acp.example.com	user@example.com	2

切换平台：

```
$ ac config use-session staging
```

```
Switched to session "staging".
```

```
Context "staging/global" activated.
```

Your First Application

让我们创建并查看一个简单的应用以验证环境是否正常：

Create a Simple Pod

1. 创建一个基础 Pod 配置：

```
cat > test-pod.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  labels:
    app: test
spec:
  containers:
    - name: nginx
      image: nginx:1.20
      ports:
        - containerPort: 80
EOF
```

2. 应用该配置：


```
$ ac apply -f test-pod.yaml  
pod/test-pod created
```

View Application Status

1. 列出 pods 查看应用状态：

```
$ ac get pods  
NAME          READY   STATUS    RESTARTS   AGE  
test-pod      1/1     Running   0           30s
```

2. 查看 pod 详细信息：

```
$ ac describe pod test-pod
```

3. 查看 pod 日志：

```
$ ac logs test-pod
```

Clean Up

完成后删除测试 pod：

```
ac delete -f test-pod.yaml
```

Getting Help

Built-in Help System

ACP CLI 提供多层次的全面帮助：

General Help

获取所有可用命令的概览：

```
ac help
```

Command-Specific Help

获取任意命令的详细帮助：

```
ac login --help
ac config --help
ac get --help
```

Resource Documentation

获取 Kubernetes 资源信息：

```
ac explain pod
ac explain deployment
ac explain service
```

Logging Out

完成工作或需要切换凭据时，使用注销命令：

```
$ ac logout
✓ Successfully logged out from 'prod' platform.

All session configurations have been removed.
To reconnect, run: ac login https://prod.acp.com --name prod
```

注销命令将：

- 从本地配置中移除身份验证令牌
- 清理该会话的所有集群和上下文条目

- 撤销 ACP 中当前使用的令牌
- 确保不会留下孤立配置

配置 ACP CLI

目录

Shell 自动补全

为 Bash 启用 Tab 自动补全

前提条件

操作步骤

为 Zsh 启用 Tab 自动补全

前提条件

操作步骤

使用 ACP CLI 访问 kubeconfig

前提条件

操作步骤

多集群配置处理

安全注意事项

Shell 自动补全

您可以为 Bash 或 Zsh shell 启用 Tab 自动补全功能。

为 **Bash** 启用 **Tab** 自动补全

安装 ACP CLI (ac) 后，您可以启用 Tab 自动补全，在按下 Tab 键时自动补全 ac 命令或提示选项。以下操作步骤演示如何为 Bash shell 启用 Tab 自动补全。

前提条件

- 您必须已安装 ACP CLI (ac) 。
- 您必须已安装 bash-completion 包。

操作步骤

1. 将 Bash 自动补全代码保存到文件：

```
$ ac completion bash > ac_bash_completion
```

2. 将文件复制到 `/etc/bash_completion.d/` 目录：

```
$ sudo cp ac_bash_completion /etc/bash_completion.d/
```

您也可以将文件保存到本地目录，并在您的 `.bashrc` 文件中通过 `source` 命令加载。

打开新终端时，Tab 自动补全即被启用。

为 Zsh 启用 Tab 自动补全

安装 ACP CLI (ac) 后，您可以启用 Tab 自动补全，在按下 Tab 键时自动补全 ac 命令或提示选项。以下操作步骤演示如何为 Zsh shell 启用 Tab 自动补全。

前提条件

您必须已安装 ACP CLI (ac) 。

操作步骤

要将 ac 的 Tab 自动补全添加到您的 `.zshrc` 文件，请运行以下命令：

```
cat >> ~/.zshrc<<EOF
autoload -Uz compinit
compinit
if [[ $commands[ac] ]]; then
    source <(ac completion zsh)
    compdef _ac ac
fi
EOF
```

打开新终端时，Tab 自动补全即被启用。

使用 ACP CLI 访问 kubeconfig

您可以使用 ACP CLI (ac) 登录 ACP 平台，并检索用于从命令行访问集群的 kubeconfig 文件。与传统的单集群 kubeconfig 导出不同，ac login 通过平台发现创建了一个全面的多集群配置。

前提条件

您拥有 ACP 平台端点访问权限和有效的认证凭证。

操作步骤

1. 运行以下命令登录您的 ACP 平台：

```
$ ac login <platform-url> --name <session-name>
```

- `<platform-url>`：ACP 平台的基础 URL（例如 <https://acp.prod.example.com> ↗）
- `<session-name>`：用户定义的该平台连接的友好名称（例如 "prod"、"staging"）

2. 登录过程会自动完成：

- 与 ACP 平台的认证
- 发现平台中所有可访问的集群

- 为所有集群创建带有 ACP 特定元数据的 kubeconfig 条目
- 设置指向 global 集群的默认上下文

3. 若要将配置导出到单独文件，运行：

```
$ ac config view --raw > kubeconfig
```

4. 设置 KUBECONFIG 环境变量指向导出的文件：

```
$ export KUBECONFIG=./kubeconfig
```

5. 使用 ac 与您的 ACP 集群交互：

```
$ ac get nodes
```

多集群配置处理

ACP CLI 登录过程创建了一个全面的 kubeconfig 结构，包括：

- 多个集群条目：平台中每个可访问集群对应一个条目
- 会话元数据：平台 URL、会话名称及集群描述存储在扩展字段中
- 统一认证：单一用户凭证条目适用于平台中所有集群
- 智能命名：使用 `acp:<session>:<cluster>` 格式的无冲突命名规范

安全注意事项

重要：导出的 kubeconfig 文件包含可访问您 ACP 平台集群的认证令牌。

- 请妥善保存该文件，设置合适的文件权限
- 切勿将 kubeconfig 文件提交到版本控制系统
- 注意令牌的过期和刷新要求
- 不同环境（prod、staging、dev）请使用不同的会话名称以保持清晰隔离

如果计划跨会话或多台机器重用导出的 kubeconfig 文件，请确保其安全存储，并定期使用 `ac config sync` 同步以保持集群列表的最新状态。

ac 和 kubectl 命令的使用

Kubernetes 命令行界面（CLI）kubectl 可用于对 Kubernetes 集群执行命令。由于 ACP 是一个兼容 Kubernetes 的平台，您可以使用随 ACP CLI 一起提供的支持的 kubectl 二进制文件，或者通过使用 ac 二进制文件获得扩展功能。

目录

ac 二进制文件

ACP 平台集成

智能资源路由

资源路由示例

额外命令

kubectl 二进制文件

ac 二进制文件

ac 二进制文件提供与 kubectl 二进制文件相同的功能，但扩展支持了 ACP 平台的原生附加功能，包括：

ACP 平台集成

ACP CLI 内置支持 ACP 的集中式、基于代理的多集群架构：

- 平台认证 - 内置登录命令，实现与 ACP 平台的安全认证

- 会话管理 - 通过 `ac login`、`ac config use-session` 和 `ac logout` 等命令实现多平台会话管理
- 增强配置 - 额外命令如 `ac config use-cluster`，使得在 ACP 多集群环境中工作更加便捷

智能资源路由

ACP CLI 会自动将平台级资源类型如 `User` 和 `Project` 路由到 `global` 集群，因为这些资源仅存在于平台级别。这允许您在任何集群上下文中访问它们，无需手动切换。所有其他资源则正常使用当前集群上下文。

资源路由示例

```
# 当前上下文指向业务集群
$ ac config current-context
prod/workload-a

# 用户请求全局资源 - ACP CLI 自动路由到 global 集群
$ ac get projects
(i) Note: Targeting global cluster for this command only, as 'projects' is a global resource.
NAME          STATUS  AGE
project-a     Active  32d
project-b     Active  18d

# 用户请求业务资源 - 在当前集群上操作
$ ac get pods
NAME                                READY  STATUS   RESTARTS  AGE
my-app-7d4f8c9b6-xyz123            1/1    Running  0          2h
```

额外命令

ACP CLI 包含简化 ACP 平台工作流的额外命令：

- `ac login` - 认证到 ACP 平台并配置多集群访问
- `ac logout` - 结束平台会话并清理配置
- `ac config get-sessions` - 列出所有配置的 ACP 平台会话
- `ac config use-session <session_name>` - 在 ACP 平台之间切换

- `ac config use-cluster <cluster_name>` - 在当前会话中切换集群
- `ac namespace` - 增强的命名空间管理，显示平台上下文
- `ac config sync` - 同步配置与平台状态

kubectl 二进制文件

kubectl 二进制文件旨在支持来自标准 Kubernetes 环境的新 ACP CLI 用户的现有工作流和脚本，或供偏好使用 kubectl CLI 的用户使用。现有的 kubectl 用户可以继续使用该二进制文件与 Kubernetes 基元交互，无需对 ACP 平台进行任何更改。

有关 kubectl 的更多信息，请参见 [kubectl documentation](#) 。

管理 CLI 配置文件

CLI 配置文件允许您为 ACP CLI (ac) 配置不同的配置文件或上下文。一个上下文由与昵称关联的用户认证信息和 ACP 平台服务器信息组成。

目录

便捷的配置管理

平台与会话管理

ac login - 认证并配置对 ACP 平台的访问

ac logout - 结束平台会话并清理配置

ac config get-sessions - 列出所有已配置的 ACP 平台会话

ac config use-session <session_name> - 切换 ACP 平台

日常操作

ac config use-cluster <cluster_name> - 在当前会话内切换集群

ac namespace - 查看当前状态并切换命名空间

ac config sync - 同步平台配置

了解 ACP CLI 配置结构

ACP CLI 增强的 kubeconfig 结构

元数据结构与组织

基于元数据的识别

命名规范

手动配置 CLI 配置文件

标准配置命令

手动操作示例

加载与合并规则

便捷的配置管理

ACP CLI 提供了增强命令，使配置管理比传统的 kubectl 操作更加简单。这些命令设计为与 ACP 的多集群环境无缝协作。

平台与会话管理

ac login - 认证并配置对 **ACP** 平台的访问

`ac login` 命令是建立与 ACP 平台连接的主要入口。它对用户进行认证，并自动配置所有必要的 kubectl 条目。

```
# 交互式登录 ACP 平台
ac login https://prod.acp.com --name prod

# 指定集群和命名空间登录
ac login https://prod.acp.com --name prod --cluster workload-a --namespace my-app

# 使用环境变量登录（用于自动化）
AC_LOGIN_PLATFORM_URL=https://prod.acp.com AC_LOGIN_SESSION=prod AC_LOGIN_USERNAME=user
AC_LOGIN_PASSWORD=secret ac login
```

登录流程：

1. 对 ACP 平台进行认证
2. 发现平台内所有可访问的集群
3. 在 kubectl 中创建集群和用户条目
4. 创建并激活上下文：
 - 如果指定了 `--cluster`：为该特定集群创建上下文
 - 如果指定了 `--namespace`：在上下文中设置命名空间
 - 如果未指定集群：默认为 global 集群
 - 上下文名称格式为：`<session_name>/<cluster_name>`

ac logout - 结束平台会话并清理配置

```
# 登出当前平台会话
ac logout

# 登出指定会话
ac logout --session prod
```

logout 命令会移除所有与会话相关的配置条目，包括集群、用户和上下文。

ac config get-sessions - 列出所有已配置的 **ACP** 平台会话

```
ac config get-sessions
```

示例输出：

CURRENT	SESSION	PLATFORM	USER	CLUSTERS
*	prod	https://acp.prod.example.com	user@example.com	3
	staging	https://staging.acp.example.com	user@example.com	2
	dev	https://dev.acp.example.com	dev-user@example.com	1

该命令显示：

- **CURRENT**：当前上下文是否属于该会话（用 * 标记）
- **SESSION**：会话名称（登录时用户定义）
- **PLATFORM**：基础平台 URL
- **USER**：会话认证用户名
- **CLUSTERS**：该会话中可用的集群数量

ac config use-session <session_name> - 切换 **ACP** 平台

```
# 切换到 staging 平台（默认为 global 集群）
ac config use-session staging

# 切换到会话中的特定集群
ac config use-session prod --cluster workload-a

# 指定命名空间切换
ac config use-session staging --cluster workload-b --namespace my-app
```

该命令根据您的会话和集群需求智能选择或创建合适的上下文。

日常操作

`ac config use-cluster <cluster_name>` - 在当前会话内切换集群

```
# 切换到当前会话中的 workload 集群
ac config use-cluster workload-a

# 创建带特定命名空间的新上下文
ac config use-cluster workload-b --namespace my-app
```

该命令在当前平台会话中查找或创建指定集群的上下文。

`ac namespace` - 查看当前状态并切换命名空间

查看当前状态：

```
ac namespace
```

示例输出：

You are currently in namespace "my-app-dev".

```
Context:  prod/workload-a
Cluster:  acp:prod:workload-a
Server:   https://acp.prod.example.com/kubernetes/workload-a/
Platform: https://acp.prod.example.com/
Session:  prod
```

切换命名空间：

```
ac namespace my-app-dev
```

ac config sync - 同步平台配置

```
# 同步当前平台会话
ac config sync

# 同步指定会话
ac config sync --session prod

# 同步所有会话
ac config sync --all
```

sync 命令会从 ACP 平台刷新您的配置，添加新集群并根据需要更新凭证。

了解 ACP CLI 配置结构

ACP CLI 将所有配置信息存储在标准的 `~/.kube/config` 文件中，确保与 kubectl 及其他 Kubernetes 工具完全兼容，同时增加了 ACP 特有的增强功能。

ACP CLI 增强的 kubeconfig 结构

ACP CLI 在标准 kubeconfig 格式基础上扩展了 ACP 特定的元数据，以增强平台集成：


```

apiVersion: v1
clusters:
- cluster:
    server: https://acp.prod.example.com/kubernetes/global/
    extensions:
    - name: acp.io/v1
      extension:
        isGlobal: true
        platformUrl: https://acp.prod.example.com
        sessionName: prod
        clusterName: global
        description: global cluster
        note: This cluster item is managed by ac CLI, to avoid unexpected behavior, do
not edit this item.
      name: acp:prod:global
- cluster:
    server: https://acp.prod.example.com/kubernetes/workload-a/
    extensions:
    - name: acp.io/v1
      extension:
        isGlobal: false
        platformUrl: https://acp.prod.example.com
        sessionName: prod
        clusterName: workload-a
        description: business cluster for team alpha
        note: This cluster item is managed by ac CLI, to avoid unexpected behavior, do
not edit this item.
      name: acp:prod:workload-a
contexts:
- context:
    cluster: acp:prod:global
    namespace: default
    user: acp:prod:user
    name: prod/global
- context:
    cluster: acp:prod:workload-a
    namespace: my-app
    user: acp:prod:user
    name: prod/workload-a
current-context: prod/global
kind: Config
preferences: {}
users:

```

```
- name: acp:prod:user
  user:
    token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
    extensions:
      - name: acp.io/v1
        extension:
          platformUrl: https://acp.prod.example.com
          sessionName: prod
          username: user@example.com
          note: This user item is managed by ac CLI, to avoid unexpected behavior, do not
edit this item.
```

元数据结构与组织

ACP CLI 使用扩展元数据来组织和识别配置条目：

基于元数据的识别

- 平台识别：使用 `platformUrl` 识别所属平台
- 会话关联：使用 `sessionName` 将相关集群、用户和上下文分组
- **global** 集群检测：使用 `isGlobal` 字段识别管理集群
- 用户凭证定位：在用户扩展中匹配 `sessionName` 和 `platformUrl`

命名规范

ACP CLI 创建新条目时采用统一命名规范：

- 集群条目：`acp:<session_name>:<cluster_name>`（例如 `acp:prod:global`）
- 用户条目：`acp:<session_name>:user`（例如 `acp:prod:user`）
- 上下文条目：`<session_name>/<cluster_name>`（例如 `prod/global`）

NOTE

`acp:` 前缀确保 ACP CLI 管理的条目不会与现有 kubeconfig 条目冲突。用户可以手动重命名这些条目——ACP CLI 通过元数据识别，而非名称。

手动配置 CLI 配置文件

对于需要精确控制配置的高级用户，ACP CLI 支持所有标准的 `kubectl config` 命令以进行手动 `kubeconfig` 管理。

TIP

大多数用户应使用上述便捷命令。

手动配置命令适用于高级场景：

- 自定义上下文命名 - 创建不遵循 ACP CLI 命名规范的上下文
- 非 **ACP** 环境 - 管理传统 `kubectl` 上下文与 ACP 会话并存
- 复杂多上下文场景 - 需要精确上下文控制的高级工作流
- 排查配置问题 - 调试或修复配置问题

标准配置命令

ACP CLI 完全兼容 `kubectl config` 子命令：

子命令	用法
<code>set-cluster</code>	在 CLI 配置文件中设置集群条目
<code>set-context</code>	在 CLI 配置文件中设置上下文条目
<code>use-context</code>	使用指定的上下文名称设置当前上下文
<code>set</code>	在 CLI 配置文件中设置单个值
<code>unset</code>	在 CLI 配置文件中取消设置单个值
<code>view</code>	显示当前合并的 CLI 配置

手动操作示例

创建自定义上下文：

```
# 创建自定义命名的上下文
ac config set-context my-custom-context --cluster=acp:prod:workload-a --namespace=my-app

# 切换到自定义上下文
ac config use-context my-custom-context
```

查看当前配置：

```
# 显示合并后的配置
ac config view

# 显示指定文件的配置
ac config view --config=/path/to/config
```

更新上下文命名空间：

```
# 设置当前上下文的命名空间
ac config set-context `ac config current-context` --namespace=my-namespace
```

加载与合并规则

在执行 CLI 操作时，您可以遵循以下规则来确定 CLI 配置的加载和合并顺序：

- CLI 配置文件从您的工作站加载，遵循以下层级和合并规则：
 - 如果设置了 `--config` 选项，则仅加载该文件。该标志只设置一次，不进行合并。
 - 如果设置了 `$KUBECONFIG` 环境变量，则使用该变量。该变量可以是路径列表，路径会被合并。当某个值被修改时，修改发生在定义该段的文件中。创建新值时，创建于第一个存在的文件中。如果链中无文件存在，则创建列表中的最后一个文件。
 - 否则，使用 `~/.kube/config` 文件，不进行合并。
- 使用的上下文根据以下流程中第一个匹配项确定：
 - `--context` 选项的值
 - CLI 配置文件中的 `current-context` 值

- 此阶段允许为空值
- 使用的用户和集群确定。此时您可能有或没有上下文；它们基于以下流程中第一个匹配项构建，分别运行一次用于用户和集群：
 - 用户名的 `--user` 选项和集群名的 `--cluster` 选项
 - 如果存在 `--context` 选项，则使用该上下文的价值
 - 此阶段允许为空值
- 实际使用的集群信息确定。此时您可能有或没有集群信息。集群信息的每个部分基于以下流程中第一个匹配项构建：
 - 以下命令行选项的值：`--server`、`--api-version`、`--certificate-authority`、`--insecure-skip-tls-verify`
 - 如果集群信息中存在该属性值，则使用它
 - 如果没有服务器地址，则报错
- 实际使用的用户信息确定。用户构建规则与集群相同，但每个用户只能有一种认证方式；认证方式冲突会导致操作失败。命令行选项优先于配置文件值。有效的命令行选项包括：
 - `--auth-path`
 - `--client-certificate`
 - `--client-key`
 - `--token`
- 对于仍缺失的信息，使用默认值并提示输入额外信息。

使用插件扩展 ACP CLI

您可以编写并安装插件，以扩展默认的 `ac` 命令，从而使您能够通过 ACP CLI 和 ACP 平台集成执行新的、更复杂的任务。

目录

编写 CLI 插件

- 创建一个简单插件

- 插件开发要求

- 其他资源

安装和使用 CLI 插件

- 前提条件

- 安装操作步骤

编写 CLI 插件

您可以使用任何支持编写命令行命令的编程语言或脚本为 ACP CLI (`ac`) 编写插件。请注意，插件不能覆盖已有的 `ac` 命令。

创建一个简单插件

此操作步骤创建一个简单的 Bash 插件，当执行 `ac foo` 命令时，会在终端打印一条消息。

操作步骤

1. 创建一个名为 `ac-foo` 的文件。命名插件文件时，请注意以下几点：

- 文件必须以 `ac-` 或 `kubectl-` 开头，才能被识别为插件
- 文件名决定调用插件的命令。例如，文件名为 `ac-foo-bar` 的插件可以通过命令 `ac foo bar` 调用
- 如果希望命令中包含连字符，也可以使用下划线。例如，文件名为 `ac-foo_bar` 的插件可以通过命令 `ac foo-bar` 调用

2. 将以下内容添加到该文件：

```
#!/bin/bash

# 可选参数处理
if [[ "$1" == "version" ]]; then
    echo "1.0.0"
    exit 0
fi

# 可选参数处理
if [[ "$1" == "config" ]]; then
    echo $KUBECONFIG
    exit 0
fi

echo "I am a plugin named ac-foo"
```

安装此插件后，可以通过 `ac foo` 命令调用它。

插件开发要求

- 编程语言：使用任何支持命令行接口的编程语言或脚本
- 命名规范：插件文件必须遵循 `ac-<plugin-name>` 或 `kubectl-<plugin-name>` 命名模式
- 可执行权限：插件文件必须具有可执行权限
- 命令覆盖：插件不能覆盖已有的 ACP CLI 命令
- 参数处理：插件应适当处理标准命令行参数和标志

其他资源

- 查看 `kubectl` 插件开发指南，了解实现模式和最佳实践
- 使用 Go 语言的 CLI 运行时工具进行插件开发
- 设计与集群资源交互的插件时，考虑 ACP 平台集成

安装和使用 CLI 插件

编写自定义 ACP CLI 插件后，必须先安装插件才能使用。

前提条件

- 已安装 ACP CLI (`ac`)
- 拥有以 `ac-` 或 `kubectl-` 开头的 CLI 插件文件

安装操作步骤

1. 如有必要，更新插件文件为可执行：

```
chmod +x <plugin_file>
```

2. 将文件放置在 PATH 中的任意位置，例如 `/usr/local/bin/`：

```
sudo mv <plugin_file> /usr/local/bin/
```

3. 运行 `ac plugin list`，确保插件已列出：

```
ac plugin list
```

示例输出

The following compatible plugins are available:

```
/usr/local/bin/<plugin_file>
```

如果插件未列出，请确认文件名以 `ac-` 或 `kubectl-` 开头，具有可执行权限，并且在 PATH 中。

4. 调用插件引入的新命令或选项。

例如，如果您构建并安装了 `ac-ns` 插件，可以使用以下命令查看当前命名空间：

```
ac ns
```

请注意，调用插件的命令取决于插件文件名。例如，文件名为 `ac-foo-bar` 的插件通过 `ac foo bar` 命令调用。

AC CLI 开发者命令参考

本参考提供了 AC CLI 开发者命令的描述和示例命令。有关管理员命令，请参见 [AC CLI 管理员命令参考](#)。

运行 `ac help` 可列出所有命令，或运行 `ac <command> --help` 获取特定命令的详细信息。

目录

`ac annotate`

示例用法

`ac api-resources`

示例用法

`ac api-versions`

示例用法

`ac apply`

示例用法

`ac apply edit-last-applied`

示例用法

`ac apply set-last-applied`

示例用法

`ac apply view-last-applied`

示例用法

`ac attach`

示例用法

`ac auth`

`ac auth can-i`

示例用法

ac auth reconcile

示例用法

ac auth whoami

示例用法

ac autoscale

示例用法

ac cluster-info

示例用法

ac cluster-info dump

示例用法

ac completion

示例用法

ac config

ac config current-context

示例用法

ac config delete-cluster

示例用法

ac config delete-context

示例用法

ac config delete-user

示例用法

ac config get-clusters

示例用法

ac config get-contexts

示例用法

ac config get-sessions

示例用法

ac config get-users

示例用法

ac config rename-context

示例用法

ac config set

示例用法

ac config set-cluster

示例用法

ac config set-context

示例用法

ac config set-credentials

示例用法

ac config sync

示例用法

ac config unset

示例用法

ac config use-cluster

示例用法

ac config use-context

示例用法

ac config use-session

示例用法

ac config view

示例用法

ac cp

示例用法

ac create

示例用法

ac create clusterrole

示例用法

ac create clusterrolebinding

示例用法

ac create configmap

示例用法

ac create cronjob

示例用法

ac create deployment

示例用法

ac create ingress

示例用法

ac create job

示例用法

ac create namespace

示例用法

ac create poddisruptionbudget

示例用法

ac create priorityclass

示例用法

ac create quota

示例用法

ac create role

示例用法

ac create rolebinding

示例用法

ac create secret

ac create secret docker-registry

示例用法

ac create secret generic

示例用法

ac create secret tls

示例用法

ac create service

ac create service clusterip

示例用法

ac create service externalname

示例用法

ac create service loadbalancer

示例用法

ac create service nodeport

示例用法

ac create serviceaccount

示例用法

ac create token

示例用法

ac delete

示例用法

ac describe

示例用法

ac diff

示例用法

ac edit

示例用法

ac events

示例用法

ac exec

示例用法

ac explain

示例用法

ac expose

示例用法

ac get

示例用法

ac kustomize

示例用法

ac label

示例用法

ac login

示例用法

ac logout

示例用法

ac logs

示例用法

ac namespace

示例用法

ac patch

示例用法

ac plugin

示例用法

ac plugin list

示例用法

ac port-forward

示例用法

ac process

示例用法

ac proxy

示例用法

ac replace

示例用法

ac rollout

示例用法

ac rollout history

示例用法

ac rollout pause

示例用法

ac rollout restart

示例用法

ac rollout resume

示例用法

ac rollout status

示例用法

ac rollout undo

示例用法

ac run

示例用法

ac scale

示例用法

ac set

ac set env

示例用法

ac set image

示例用法

ac set resources

示例用法

ac set selector

示例用法

ac set serviceaccount

示例用法

ac set subject

示例用法

ac version

示例用法

ac wait

示例用法

ac annotate

更新资源上的注解

示例用法

```
# 更新 pod 'foo', 添加注解 'description', 值为 'my frontend'
# 如果同一注解被多次设置, 只有最后一个值生效
ac annotate pods foo description='my frontend'

# 更新通过类型和名称标识的 pod, 配置文件为 "pod.json"
ac annotate -f pod.json description='my frontend'

# 更新 pod 'foo', 添加注解 'description', 值为 'my frontend running nginx', 覆盖已有值
ac annotate --overwrite pods foo description='my frontend running nginx'

# 更新命名空间中的所有 pod
ac annotate pods --all description='my frontend running nginx'

# 仅当资源版本为 1 时更新 pod 'foo'
ac annotate pods foo description='my frontend running nginx' --resource-version=1

# 如果存在, 删除 pod 'foo' 上名为 'description' 的注解
# 不需要 --overwrite 标志
ac annotate pods foo description-
```

ac api-resources

打印服务器支持的 API 资源

示例用法

```
# 打印支持的 API 资源
ac api-resources

# 打印支持的 API 资源及更多信息
ac api-resources -o wide

# 按列排序打印支持的 API 资源
ac api-resources --sort-by=name

# 打印支持的命名空间资源
ac api-resources --namespaced=true

# 打印支持的非命名空间资源
ac api-resources --namespaced=false

# 打印指定 APIGroup 的支持资源
ac api-resources --api-group=rbac.authorization.k8s.io
```

ac api-versions

打印服务器支持的 API 版本，格式为 "group/version"

示例用法

```
# 打印支持的 API 版本
ac api-versions
```

ac apply

通过文件名或标准输入应用配置到资源

示例用法

```
# 应用 pod.json 中的配置到 pod
ac apply -f ./pod.json

# 应用包含 kustomization.yaml 的目录资源，例如 dir/kustomization.yaml
ac apply -k dir/

# 从标准输入传入 JSON 并应用到 pod
cat pod.json | ac apply -f -

# 应用所有以 '.json' 结尾的文件中的配置
ac apply -f '*.json'

# 注意：--prune 仍处于 Alpha 阶段
# 应用 manifest.yaml 中匹配标签 app=nginx 的配置，并删除文件中未包含且匹配标签 app=nginx
的其他资源
ac apply --prune -f manifest.yaml -l app=nginx

# 应用 manifest.yaml 中的配置，并删除文件中未包含的所有 ConfigMap
ac apply --prune -f manifest.yaml --all --prune-allowlist=core/v1/ConfigMap
```

ac apply edit-last-applied

编辑资源/对象的最新 last-applied-configuration 注解

示例用法

```
# 以 YAML 格式编辑指定类型/名称的 last-applied-configuration 注解
ac apply edit-last-applied deployment/nginx

# 以 JSON 格式编辑文件中的 last-applied-configuration 注解
ac apply edit-last-applied -f deploy.yaml -o json
```

ac apply set-last-applied

设置 live 对象的 last-applied-configuration 注解，使其匹配文件内容

示例用法

```
# 设置资源的 last-applied-configuration 以匹配文件内容
ac apply set-last-applied -f deploy.yaml

# 针对目录中的每个配置文件执行 set-last-applied
ac apply set-last-applied -f path/

# 设置资源的 last-applied-configuration, 若注解不存在则创建
ac apply set-last-applied -f deploy.yaml --create-annotation=true
```

ac apply view-last-applied

查看资源/对象的最新 last-applied-configuration 注解

示例用法

```
# 以 YAML 格式查看指定类型/名称的 last-applied-configuration 注解
ac apply view-last-applied deployment/nginx

# 以 JSON 格式查看文件中的 last-applied-configuration 注解
ac apply view-last-applied -f deploy.yaml -o json
```

ac attach

附加到正在运行的容器

示例用法

```
# 获取运行中 pod mypod 的输出；使用 'ac.kubernetes.io/default-container' 注解选择容器，否则选择第一个容器
ac attach mypod

# 获取 pod mypod 中 ruby-container 的输出
ac attach mypod -c ruby-container

# 切换到原始终端模式；将 stdin 发送到 pod mypod 中 ruby-container 的 'bash'，并将 stdout/stderr 返回客户端
ac attach mypod -c ruby-container -i -t

# 获取名为 nginx 的 replica set 的第一个 pod 的输出
ac attach rs/nginx
```

ac auth

检查授权

ac auth can-i

检查是否允许执行某个操作

示例用法

```
# 检查是否可以在所有命名空间创建 pods
ac auth can-i create pods --all-namespaces

# 检查是否可以在当前命名空间列出 deployments
ac auth can-i list deployments.apps

# 检查命名空间 "dev" 中的服务账户 "foo" 是否可以在命名空间 "prod" 列出 pods
# 需要允许使用全局选项 "--as" 的模拟
ac auth can-i list pods --as=system:serviceaccount:dev:foo -n prod

# 检查是否可以在当前命名空间执行所有操作（"*" 表示全部）
ac auth can-i '*' '*'

# 检查是否可以获取命名空间 "foo" 中名为 "bar" 的 job
ac auth can-i list jobs.batch/bar -n foo

# 检查是否可以读取 pod 日志
ac auth can-i get pods --subresource=log

# 检查是否可以访问 URL /logs/
ac auth can-i get /logs/

# 检查是否可以批准 certificates.k8s.io
ac auth can-i approve certificates.k8s.io

# 列出命名空间 "foo" 中所有允许的操作
ac auth can-i --list --namespace=foo
```

ac auth reconcile

对 RBAC 角色、角色绑定、集群角色和集群角色绑定对象进行规则对齐

示例用法

```
# 从文件对齐 RBAC 资源
ac auth reconcile -f my-rbac-rules.yaml
```

ac auth whoami

实验性功能：检查自身主体属性

示例用法

```
# 获取你的主体属性
ac auth whoami

# 以 JSON 格式获取你的主体属性
ac auth whoami -o json
```

ac autoscale

自动扩缩部署、ReplicaSet、StatefulSet 或 ReplicationController

示例用法

```
# 自动扩缩部署 "foo", pod 数量在 2 到 10 之间, 未指定目标 CPU 利用率, 使用默认自动扩缩策略
ac autoscale deployment foo --min=2 --max=10

# 自动扩缩 ReplicationController "foo", pod 数量在 1 到 5 之间, 目标 CPU 利用率为 80%
ac autoscale rc foo --max=5 --cpu-percent=80
```

ac cluster-info

显示集群信息

示例用法

```
# 打印控制平面和集群服务地址
ac cluster-info
```

ac cluster-info dump

导出调试和诊断相关信息

示例用法

```
# 导出当前集群状态到标准输出
ac cluster-info dump

# 导出当前集群状态到 /path/to/cluster-state
ac cluster-info dump --output-directory=/path/to/cluster-state

# 导出所有命名空间到标准输出
ac cluster-info dump --all-namespaces

# 导出指定命名空间集合到 /path/to/cluster-state
ac cluster-info dump --namespaces default,kube-system --output-
directory=/path/to/cluster-state
```

ac completion

输出指定 shell (bash、zsh、fish 或 powershell) 的补全代码

示例用法


```

# 在 macOS 使用 homebrew 安装 bash 补全
## 如果使用 macOS 自带的 Bash 3.2
brew install bash-completion
## 或者使用 Bash 4.1+
brew install bash-completion@2
## 如果通过 homebrew 安装 ac, 补全应立即生效
## 如果通过其他方式安装, 可能需要将补全脚本添加到补全目录
ac completion bash > $(brew --prefix)/etc/bash_completion.d/ac

# 在 Linux 安装 bash 补全
## 如果未安装 bash-completion, 请通过发行版包管理器安装 'bash-completion' 包
## 加载 ac 的 bash 补全代码到当前 shell
source <(ac completion bash)
## 将 bash 补全代码写入文件并在 .bash_profile 中 source
ac completion bash > ~/.kube/completion.bash.inc
printf "
# ac shell completion
source '$HOME/.kube/completion.bash.inc'
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# 加载 zsh[1] 的 ac 补全代码到当前 shell
source <(ac completion zsh)
# 设置 zsh[1] 的 ac 补全代码开机自动加载
ac completion zsh > "${fpath[1]}/_ac"

# 加载 fish[2] 的 ac 补全代码到当前 shell
ac completion fish | source
# 要在每个会话加载补全, 执行一次:
ac completion fish > ~/.config/fish/completions/ac.fish

# 加载 powershell 的 ac 补全代码到当前 shell
ac completion powershell | Out-String | Invoke-Expression
# 设置 powershell 补全代码开机自动运行
## 保存补全代码到脚本并在配置文件中执行
ac completion powershell > $HOME\.kube\completion.ps1
Add-Content $PROFILE "$HOME\.kube\completion.ps1"
## 在配置文件中执行补全代码
Add-Content $PROFILE "if (Get-Command ac -ErrorAction SilentlyContinue) {
ac completion powershell | Out-String | Invoke-Expression
}"

```

```
## 直接将补全代码添加到 $PROFILE 脚本  
ac completion powershell >> $PROFILE
```

ac config

修改 kubeconfig 文件

ac config current-context

显示当前上下文

示例用法

```
# 显示当前上下文  
ac config current-context
```

ac config delete-cluster

从 kubeconfig 中删除指定集群

示例用法

```
# 删除 minikube 集群  
ac config delete-cluster minikube
```

ac config delete-context

从 kubeconfig 中删除指定上下文

示例用法

```
# 删除 minikube 集群的上下文  
ac config delete-context minikube
```

ac config delete-user

从 kubeconfig 中删除指定用户

示例用法

```
# 删除 minikube 用户  
ac config delete-user minikube
```

ac config get-clusters

显示 kubeconfig 中定义的集群

示例用法

```
# 列出 ac 已知的集群  
ac config get-clusters
```

ac config get-contexts

描述一个或多个上下文

示例用法

```
# 列出 kubeconfig 文件中的所有上下文
ac config get-contexts
```

```
# 描述 kubeconfig 文件中的一个上下文
ac config get-contexts my-context
```

ac config get-sessions

列出所有配置的 ACP 平台会话

示例用法

```
# 列出所有配置的 ACP 会话
ac config get-sessions
```

ac config get-users

显示 kubeconfig 中定义的用户

示例用法

```
# 列出 ac 已知的用户
ac config get-users
```

ac config rename-context

重命名 kubeconfig 文件中的上下文

示例用法

```
# 将上下文 'old-name' 重命名为 'new-name'  
ac config rename-context old-name new-name
```

ac config set

设置 kubeconfig 文件中的单个值

示例用法

```
# 将 my-cluster 集群的 server 字段设置为 https://1.2.3.4  
ac config set clusters.my-cluster.server https://1.2.3.4  
  
# 设置 my-cluster 集群的 certificate-authority-data 字段  
ac config set clusters.my-cluster.certificate-authority-data $(echo "cert_data_here" |  
base64 -i -)  
  
# 将 my-context 上下文的 cluster 字段设置为 my-cluster  
ac config set contexts.my-context.cluster my-cluster  
  
# 使用 --set-raw-bytes 选项设置 cluster-admin 用户的 client-key-data 字段  
ac config set users.cluster-admin.client-key-data cert_data_here --set-raw-bytes=true
```

ac config set-cluster

设置 kubeconfig 中的集群条目

示例用法

```
# 仅设置 e2e 集群条目的 server 字段, 不修改其他值
ac config set-cluster e2e --server=https://1.2.3.4

# 为 e2e 集群条目嵌入证书授权数据
ac config set-cluster e2e --embed-certs --certificate-
authority=~/.kube/e2e/kubernetes.ca.crt

# 禁用 e2e 集群条目的证书校验
ac config set-cluster e2e --insecure-skip-tls-verify=true

# 设置 e2e 集群条目的自定义 TLS 服务器名称用于验证
ac config set-cluster e2e --tls-server-name=my-cluster-name

# 设置 e2e 集群条目的代理 URL
ac config set-cluster e2e --proxy-url=https://1.2.3.4
```

ac config set-context

设置 kubeconfig 中的上下文条目

示例用法

```
# 设置 gce 上下文条目的用户字段, 不修改其他值
ac config set-context gce --user=cluster-admin
```

ac config set-credentials

设置 kubeconfig 中的用户条目

示例用法

```
# 仅设置 cluster-admin 条目的 "client-key" 字段, 不修改其他值
ac config set-credentials cluster-admin --client-key=~/.kube/admin.key

# 设置 cluster-admin 条目的基本认证
ac config set-credentials cluster-admin --username=admin --password=uXFGweU9l35qcif

# 在 cluster-admin 条目中嵌入客户端证书数据
ac config set-credentials cluster-admin --client-certificate=~/.kube/admin.crt --embed-
certs=true

# 启用 cluster-admin 条目的 Google Compute Platform 认证提供者
ac config set-credentials cluster-admin --auth-provider=gcp

# 启用 cluster-admin 条目的 OpenID Connect 认证提供者并添加参数
ac config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-
id=foo --auth-provider-arg=client-secret=bar

# 移除 cluster-admin 条目中 OpenID Connect 认证提供者的 "client-secret" 配置值
ac config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-
secret-

# 启用 cluster-admin 条目的新 exec 认证插件
ac config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-
api-version=client.authentication.k8s.io/v1beta1

# 启用 cluster-admin 条目的新 exec 认证插件并开启交互模式
ac config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-
api-version=client.authentication.k8s.io/v1beta1 --exec-interactive-mode=Never

# 为 cluster-admin 条目定义新的 exec 认证插件参数
ac config set-credentials cluster-admin --exec-arg=arg1 --exec-arg=arg2

# 为 cluster-admin 条目创建或更新 exec 认证插件环境变量
ac config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=key2=val2

# 移除 cluster-admin 条目的 exec 认证插件环境变量
ac config set-credentials cluster-admin --exec-env=var-to-remove-
```

ac config sync

同步 kubeconfig 与 ACP 平台状态

示例用法

```
# 基于当前上下文同步会话
ac config sync

# 同步指定会话
ac config sync --session prod

# 同步所有会话
ac config sync --all
```

ac config unset

取消设置 kubeconfig 文件中的单个值

示例用法

```
# 取消设置 current-context
ac config unset current-context

# 取消设置 foo 上下文中的 namespace
ac config unset contexts.foo.namespace
```

ac config use-cluster

通过集群名称切换到指定 ACP 集群

示例用法


```
# 切换到 workload-a 集群的现有上下文
ac config use-cluster workload-a

# 为 workload-b 创建带命名空间的新上下文
ac config use-cluster workload-b --namespace my-app

# 切换到 global 集群
ac config use-cluster global
```

ac config use-context

设置 kubeconfig 文件中的 current-context

示例用法

```
# 使用 minikube 集群的上下文
ac config use-context minikube
```

ac config use-session

切换到指定的 ACP 会话并智能选择上下文

示例用法

```
# 切换到 staging 会话（默认 global 集群）
ac config use-session staging

# 切换到指定集群的生产会话
ac config use-session prod --cluster workload-b

# 切换到指定集群和命名空间的会话
ac config use-session staging --cluster workload-a --namespace my-app
```

ac config view

显示合并的 kubeconfig 设置或指定的 kubeconfig 文件

示例用法

```
# 显示合并的 kubeconfig 设置
```

```
ac config view
```

```
# 显示合并的 kubeconfig 设置，包含原始证书数据和暴露的密钥
```

```
ac config view --raw
```

```
# 获取 e2e 用户的密码
```

```
ac config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

ac cp

在容器和本地之间复制文件和目录

示例用法

```

# !!!重要提示!!!
# 需要容器镜像中存在 'tar' 二进制文件
# 如果容器中没有 'tar', 'ac cp' 将失败
#
# 对于高级用例, 如符号链接、通配符扩展或文件模式保留, 建议使用 'ac exec'

# 将本地文件 /tmp/foo 复制到命名空间 <some-namespace> 中远程 pod 的 /tmp/bar
tar cf - /tmp/foo | ac exec -i -n <some-namespace> <some-pod> -- tar xf - -C /tmp/bar

# 将远程 pod 的 /tmp/foo 复制到本地 /tmp/bar
ac exec -n <some-namespace> <some-pod> -- tar cf - /tmp/foo | tar xf - -C /tmp/bar

# 将本地目录 /tmp/foo_dir 复制到默认命名空间远程 pod 的 /tmp/bar_dir
ac cp /tmp/foo_dir <some-pod>:/tmp/bar_dir

# 将本地文件 /tmp/foo 复制到远程 pod 中指定容器的 /tmp/bar
ac cp /tmp/foo <some-pod>:/tmp/bar -c <specific-container>

# 将本地文件 /tmp/foo 复制到命名空间 <some-namespace> 中远程 pod 的 /tmp/bar
ac cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar

# 将远程 pod 的 /tmp/foo 复制到本地 /tmp/bar
ac cp <some-namespace>/<some-pod>:/tmp/foo /tmp/bar

```

ac create

从文件或标准输入创建资源

示例用法

```
# 使用 pod.json 中的数据创建 pod
ac create -f ./pod.json

# 基于标准输入传入的 JSON 创建 pod
cat pod.json | ac create -f -

# 编辑 registry.yaml 中的数据（JSON 格式），然后使用编辑后的数据创建资源
ac create -f registry.yaml --edit -o json
```

ac create clusterrole

创建集群角色

示例用法

```
# 创建名为 "pod-reader" 的集群角色，允许用户对 pods 执行 "get"、"watch" 和 "list"
ac create clusterrole pod-reader --verb=get,list,watch --resource=pods

# 创建名为 "pod-reader" 的集群角色，指定 ResourceName
ac create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod -
-resource-name=anotherpod

# 创建名为 "foo" 的集群角色，指定 API Group
ac create clusterrole foo --verb=get,list,watch --resource=rs.apps

# 创建名为 "foo" 的集群角色，指定 SubResource
ac create clusterrole foo --verb=get,list,watch --resource=pods,pods/status

# 创建名为 "foo" 的集群角色，指定 NonResourceURL
ac create clusterrole "foo" --verb=get --non-resource-url=/logs/*

# 创建名为 "monitoring" 的集群角色，指定 AggregationRule
ac create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-
monitoring=true"
```

ac create clusterrolebinding

为特定集群角色创建集群角色绑定

示例用法

```
# 使用 cluster-admin 集群角色为 user1、user2 和 group1 创建集群角色绑定
ac create clusterrolebinding cluster-admin --clusterrole=cluster-admin --user=user1 --
user=user2 --group=group1
```

ac create configmap

从本地文件、目录或字面值创建 config map

示例用法

```
# 基于目录 bar 创建名为 my-config 的新 config map
ac create configmap my-config --from-file=path/to/bar

# 创建名为 my-config 的 config map, 指定键而非文件名
ac create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-
file=key2=/path/to/bar/file2.txt

# 创建名为 my-config 的 config map, 包含 key1=config1 和 key2=config2
ac create configmap my-config --from-literal=key1=config1 --from-literal=key2=config2

# 从文件中的键值对创建名为 my-config 的 config map
ac create configmap my-config --from-file=path/to/bar

# 从 env 文件创建名为 my-config 的 config map
ac create configmap my-config --from-env-file=path/to/foo.env --from-env-
file=path/to/bar.env
```

ac create cronjob

创建指定名称的定时任务

示例用法

```
# 创建定时任务
ac create cronjob my-job --image=busybox --schedule="*/1 * * * *"

# 创建带命令的定时任务
ac create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

ac create deployment

创建指定名称的部署

示例用法

```
# 创建名为 my-dep 的部署，运行 busybox 镜像
ac create deployment my-dep --image=busybox

# 创建带命令的部署
ac create deployment my-dep --image=busybox -- date

# 创建名为 my-dep 的部署，运行 nginx 镜像，副本数为 3
ac create deployment my-dep --image=nginx --replicas=3

# 创建名为 my-dep 的部署，运行 busybox 镜像，暴露端口 5701
ac create deployment my-dep --image=busybox --port=5701

# 创建名为 my-dep 的部署，运行多个容器
ac create deployment my-dep --image=busybox:latest --image=ubuntu:latest --image=nginx
```

ac create ingress

创建指定名称的 ingress

示例用法

```

# 创建名为 'simple' 的单 ingress, 将 foo.com/bar 的请求转发到 svc1:8080, 使用 TLS 证书
"my-cert"
ac create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

# 创建 catch-all ingress, 路径为 "/path", 指向服务 svc:port, Ingress 类为 "otheringress"
ac create ingress catch-all --class=otheringress --rule="/path=svc:port"

# 创建带两个注解的 ingress: ingress.annotation1 和 ingress.annotation2
ac create ingress annotated --class=default --rule="foo.com/bar=svc:port" \
--annotation ingress.annotation1=foo \
--annotation ingress.annotation2=bla

# 创建同一主机多个路径的 ingress
ac create ingress multipath --class=default \
--rule="foo.com/=svc:port" \
--rule="foo.com/admin/=svcadmin:portadmin"

# 创建多主机且 pathType 为 Prefix 的 ingress
ac create ingress ingress1 --class=default \
--rule="foo.com/path*=svc:8080" \
--rule="bar.com/admin*=svc2:http"

# 创建启用 TLS 的 ingress, 使用默认 ingress 证书和不同路径类型
ac create ingress ingtls --class=default \
--rule="foo.com/=svc:https,tls" \
--rule="foo.com/path/subpath*=othersvc:8080"

# 创建启用 TLS 的 ingress, 使用指定 secret, pathType 为 Prefix
ac create ingress ingsecret --class=default \
--rule="foo.com/*=svc:8080,tls=secret1"

# 创建带默认后端的 ingress
ac create ingress ingdefault --class=default \
--default-backend=defaultsvc:http \
--rule="foo.com/*=svc:8080,tls=secret1"

```

ac create job

创建指定名称的 job

示例用法

```
# 创建 job
ac create job my-job --image=busybox

# 创建带命令的 job
ac create job my-job --image=busybox -- date

# 从名为 "a-cronjob" 的 cronjob 创建 job
ac create job test-job --from=cronjob/a-cronjob
```

ac create namespace

创建指定名称的命名空间

示例用法

```
# 创建名为 my-namespace 的新命名空间
ac create namespace my-namespace
```

ac create poddisruptionbudget

创建指定名称的 pod 中断预算

示例用法

```
# 创建名为 my-pdb 的 pod 中断预算，选择所有带有 app=rails 标签的 pods，要求至少有一个可用
ac create poddisruptionbudget my-pdb --selector=app=rails --min-available=1

# 创建名为 my-pdb 的 pod 中断预算，选择所有带有 app=nginx 标签的 pods，要求至少半数可用
ac create pdb my-pdb --selector=app=nginx --min-available=50%
```

ac create priorityclass

创建指定名称的优先级类

示例用法

```
# 创建名为 high-priority 的优先级类
ac create priorityclass high-priority --value=1000 --description="high priority"

# 创建名为 default-priority 的优先级类，作为全局默认优先级
ac create priorityclass default-priority --value=1000 --global-default=true --
description="default priority"

# 创建名为 high-priority 的优先级类，禁止抢占优先级较低的 pods
ac create priorityclass high-priority --value=1000 --description="high priority" --
preemption-policy="Never"
```

ac create quota

创建指定名称的资源配额

示例用法

```
# 创建名为 my-quota 的新资源配额
ac create quota my-quota --
hard=cpu=1,memory=1G,pods=2,services=3,replicationcontrollers=2,resourcequotas=1,secrets=5,p

# 创建名为 best-effort 的新资源配额
ac create quota best-effort --hard=pods=100 --scopes=BestEffort
```

ac create role

创建单规则角色

示例用法

```
# 创建名为 "pod-reader" 的角色, 允许用户对 pods 执行 "get"、"watch" 和 "list"
ac create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

# 创建名为 "pod-reader" 的角色, 指定 ResourceName
ac create role pod-reader --verb=get --resource=pods --resource-name=readablepod --
resource-name=anotherpod

# 创建名为 "foo" 的角色, 指定 API Group
ac create role foo --verb=get,list,watch --resource=rs.apps

# 创建名为 "foo" 的角色, 指定 SubResource
ac create role foo --verb=get,list,watch --resource=pods,pods/status
```

ac create rolebinding

为特定角色或集群角色创建角色绑定

示例用法

```
# 使用 admin 集群角色为 user1、user2 和 group1 创建角色绑定
ac create rolebinding admin --clusterrole=admin --user=user1 --user=user2 --group=group1

# 使用 admin 角色为服务账户 monitoring:sa-dev 创建角色绑定
ac create rolebinding admin-binding --role=admin --serviceaccount=monitoring:sa-dev
```

ac create secret

使用指定子命令创建 secret

ac create secret docker-registry

创建用于 Docker registry 的 secret

示例用法

```
# 如果没有 .dockercfg 文件, 直接创建 dockercfg secret
ac create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --
docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL

# 从 ~/.docker/config.json 创建名为 my-secret 的新 secret
ac create secret docker-registry my-secret --from-file=path/to/.docker/config.json
```

ac create secret generic

从本地文件、目录或字面值创建 secret

示例用法

```
# 创建名为 my-secret 的新 secret, 包含目录 bar 中每个文件的键
ac create secret generic my-secret --from-file=path/to/bar

# 创建名为 my-secret 的 secret, 指定键而非文件名
ac create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-
file=ssh-publickey=path/to/id_rsa.pub

# 创建名为 my-secret 的 secret, 包含 key1=supersecret 和 key2=topsecret
ac create secret generic my-secret --from-literal=key1=supersecret --from-
literal=key2=topsecret

# 使用文件和字面值组合创建名为 my-secret 的 secret
ac create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-
literal=passphrase=topsecret

# 从 env 文件创建名为 my-secret 的 secret
ac create secret generic my-secret --from-env-file=path/to/foo.env --from-env-
file=path/to/bar.env
```

ac create secret tls

创建 TLS secret

示例用法

```
# 创建名为 tls-secret 的 TLS secret, 指定密钥对
ac create secret tls tls-secret --cert=path/to/tls.crt --key=path/to/tls.key
```

ac create service

使用指定子命令创建服务

ac create service clusterip

创建 ClusterIP 服务

示例用法

```
# 创建名为 my-cs 的 ClusterIP 服务
ac create service clusterip my-cs --tcp=5678:8080

# 创建名为 my-cs 的 ClusterIP 服务（无头模式）
ac create service clusterip my-cs --clusterip="None"
```

ac create service externalname

创建 ExternalName 服务

示例用法

```
# 创建名为 my-ns 的 ExternalName 服务
ac create service externalname my-ns --external-name bar.com
```

ac create service loadbalancer

创建 LoadBalancer 服务

示例用法

```
# 创建名为 my-lbs 的 LoadBalancer 服务
ac create service loadbalancer my-lbs --tcp=5678:8080
```

ac create service nodeport

创建 NodePort 服务

示例用法

```
# 创建名为 my-ns 的 NodePort 服务
ac create service nodeport my-ns --tcp=5678:8080
```

ac create serviceaccount

创建指定名称的服务账户

示例用法

```
# 创建名为 my-service-account 的新服务账户
ac create serviceaccount my-service-account
```

ac create token

请求服务账户令牌

示例用法

请求当前命名空间中服务账户 "myapp" 的令牌，用于认证 kube-apiserver

```
ac create token myapp
```

请求自定义命名空间中服务账户的令牌

```
ac create token myapp --namespace myns
```

请求带自定义过期时间的令牌

```
ac create token myapp --duration 10m
```

请求带自定义受众的令牌

```
ac create token myapp --audience https://example.com
```

请求绑定到 Secret 对象实例的令牌

```
ac create token myapp --bound-object-kind Secret --bound-object-name mysecret
```

请求绑定到具有特定 UID 的 Secret 对象实例的令牌

```
ac create token myapp --bound-object-kind Secret --bound-object-name mysecret --bound-object-uid 0d4691ed-659b-4935-a832-355f77ee47cc
```

ac delete

通过文件名、标准输入、资源和名称，或资源和标签选择器删除资源

示例用法


```
# 使用 pod.json 中指定的类型和名称删除 pod
ac delete -f ./pod.json

# 从包含 kustomization.yaml 的目录删除资源, 例如 dir/kustomization.yaml
ac delete -k dir

# 删除所有以 '.json' 结尾的文件中的资源
ac delete -f '*.json'

# 基于标准输入传入的 JSON 删除 pod
cat pod.json | ac delete -f -

# 删除名称为 "baz" 和 "foo" 的 pods 和 services
ac delete pod,service baz foo

# 删除标签为 name=myLabel 的 pods 和 services
ac delete pods,services -l name=myLabel

# 立即删除 pod foo
ac delete pod foo --now

# 强制删除死节点上的 pod foo
ac delete pod foo --force

# 删除所有 pods
ac delete pods --all

# 仅在用户确认删除时删除所有 pods
ac delete pods --all --interactive
```

ac describe

显示特定资源或资源组的详细信息

示例用法

```
# 描述节点
ac describe nodes kubernetes-node-emt8.c.myproject.internal

# 描述 pod
ac describe pods/nginx

# 描述通过类型和名称标识的 pod，配置文件为 "pod.json"
ac describe -f pod.json

# 描述所有 pods
ac describe pods

# 按标签 name=myLabel 描述 pods
ac describe pods -l name=myLabel

# 描述由 'frontend' replication controller 管理的所有 pods
# （rc 创建的 pods 名称以 rc 名称为前缀）
ac describe pods frontend
```

ac diff

比较 live 版本与将要应用的版本差异

示例用法

```
# 比较 pod.json 中的资源
ac diff -f pod.json

# 比较从标准输入读取的文件
cat service.yaml | ac diff -f -
```

ac edit

编辑服务器上的资源

示例用法

```
# 编辑名为 'registry' 的服务
ac edit svc/registry

# 使用替代编辑器
KUBE_EDITOR="nano" ac edit svc/registry

# 以 JSON 格式使用 v1 API 编辑 job 'myjob'
ac edit job.v1.batch/myjob -o json

# 以 YAML 格式编辑部署 'mydeployment', 并将修改后的配置保存到注解中
ac edit deployment/mydeployment -o yaml --save-config

# 编辑 'mydeployment' 部署的 'status' 子资源
ac edit deployment mydeployment --subresource='status'
```

ac events

列出事件

示例用法

```
# 列出默认命名空间的近期事件
ac events

# 列出所有命名空间的近期事件
ac events --all-namespaces

# 列出指定 pod 的近期事件, 并持续监听新事件
ac events --for pod/web-pod-13je7 --watch

# 以 YAML 格式列出近期事件
ac events -oyaml

# 仅列出类型为 'Warning' 或 'Normal' 的近期事件
ac events --types=Warning,Normal
```

ac exec

在容器中执行命令

示例用法

```
# 从 pod mypod 的第一个容器获取 'date' 命令输出
ac exec mypod -- date

# 从 pod mypod 的 ruby-container 容器获取 'date' 命令输出
ac exec mypod -c ruby-container -- date

# 切换到原始终端模式；将 stdin 发送到 pod mypod 中 ruby-container 的 'bash'，并将
stdout/stderr 返回客户端
ac exec mypod -c ruby-container -i -t -- bash -il

# 列出 pod mypod 第一个容器的 /usr 目录内容，并按修改时间排序
# 如果命令参数与 ac exec 参数有冲突，需使用 -- 分隔
ac exec mypod -i -t -- ls -lt /usr

# 从部署 mydeployment 的第一个 pod 获取 'date' 命令输出，默认使用第一个容器
ac exec deploy/mydeployment -- date

# 从服务 myservice 的第一个 pod 获取 'date' 命令输出，默认使用第一个容器
ac exec svc/myservice -- date
```

ac explain

获取资源文档

示例用法

```
# 获取资源及其字段的文档
ac explain pods

# 获取资源中所有字段
ac explain pods --recursive

# 获取支持的 api 版本中 deployment 的说明
ac explain deployments --api-version=apps/v1

# 获取资源特定字段的文档
ac explain pods.spec.containers

# 以不同格式获取资源文档
ac explain deployment --output=plaintext-openapi2
```

ac expose

将 replication controller、service、deployment 或 pod 公开为新的 Kubernetes 服务

示例用法

```
# 为复制的 nginx 创建服务, 端口为 80, 连接容器端口 8000
ac expose rc nginx --port=80 --target-port=8000

# 为通过类型和名称指定的 replication controller 创建服务, 端口为 80, 连接容器端口 8000
ac expose -f nginx-controller.yaml --port=80 --target-port=8000

# 为 pod valid-pod 创建服务, 端口为 444, 服务名为 "frontend"
ac expose pod valid-pod --port=444 --name=frontend

# 基于上述服务创建第二个服务, 暴露容器端口 8443 为端口 443, 服务名为 "nginx-https"
ac expose service nginx --port=443 --target-port=8443 --name=nginx-https

# 为复制的流媒体应用创建服务, 端口为 4100, 负载 UDP 流量, 服务名为 'video-stream'
ac expose rc streamer --port=4100 --protocol=UDP --name=video-stream

# 为复制的 nginx 创建基于 replica set 的服务, 端口为 80, 连接容器端口 8000
ac expose rs nginx --port=80 --target-port=8000

# 为 nginx 部署创建服务, 端口为 80, 连接容器端口 8000
ac expose deployment nginx --port=80 --target-port=8000
```

ac get

显示一个或多个资源

示例用法

以 ps 输出格式列出所有 pods

```
ac get pods
```

以 ps 输出格式列出所有 pods, 显示更多信息 (如节点名称)

```
ac get pods -o wide
```

以 ps 输出格式列出指定名称的 replication controller

```
ac get replicationcontroller web
```

以 JSON 输出格式列出 "apps" API 组 "v1" 版本的 deployments

```
ac get deployments.v1.apps -o json
```

以 JSON 输出格式列出单个 pod

```
ac get -o json pod web-pod-13je7
```

以 JSON 输出格式列出通过类型和名称指定的 pod, 配置文件为 "pod.yaml"

```
ac get -f pod.yaml -o json
```

从包含 kustomization.yaml 的目录获取资源, 例如 dir/kustomization.yaml

```
ac get -k dir/
```

仅返回指定 pod 的 phase 值

```
ac get -o template pod/web-pod-13je7 --template={{.status.phase}}
```

以自定义列格式列出 pod test-pod 的资源信息

```
ac get pod test-pod -o custom-
```

```
columns=CONTAINER:.spec.containers[0].name,IMAGE:.spec.containers[0].image
```

以 ps 输出格式列出所有 replication controllers 和 services

```
ac get rc,services
```

按类型和名称列出一个或多个资源

```
ac get rc/web service/frontend pods/web-pod-13je7
```

列出单个 pod 的 'status' 子资源

```
ac get pod web-pod-13je7 --subresource status
```

列出命名空间 'backend' 中的所有 deployments

```
ac get deployments.apps --namespace backend
```

列出所有命名空间中存在的所有 pods

```
ac get pods --all-namespaces
```

ac kustomize

从目录或 URL 构建 kustomization 目标

示例用法

```
# 构建当前工作目录
ac kustomize

# 构建共享配置目录
ac kustomize /home/config/production

# 从 github 构建
ac kustomize https://github.com/kubernetes-sigs/kustomize.git/examples/helloWorld?
ref=v1.0.6
```

ac label

更新资源上的标签

示例用法


```
# 更新 pod 'foo', 添加标签 'unhealthy', 值为 'true'
ac label pods foo unhealthy=true

# 更新 pod 'foo', 添加标签 'status', 值为 'unhealthy', 覆盖已有值
ac label --overwrite pods foo status=unhealthy

# 更新命名空间中的所有 pod
ac label pods --all status=unhealthy

# 更新通过类型和名称标识的 pod, 配置文件为 "pod.json"
ac label -f pod.json status=unhealthy

# 仅当资源版本为 1 时更新 pod 'foo'
ac label pods foo status=unhealthy --resource-version=1

# 如果存在, 删除 pod 'foo' 上名为 'bar' 的标签
# 不需要 --overwrite 标志
ac label pods foo bar-
```

ac login

登录 ACP 平台

示例用法

交互式登录（提示缺失参数）

```
ac login https://example.com --name prod
```

通过参数标志登录

```
ac login https://example.com --name prod --username=myuser --password=mypassword
```

使用环境变量登录（适合自动化）

```
AC_LOGIN_PLATFORM_URL=https://example.com AC_LOGIN_SESSION=prod \  
AC_LOGIN_USERNAME=myuser AC_LOGIN_PASSWORD=mypassword ac login
```

使用特定身份提供者登录

```
ac login https://example.com --name prod --idp ldap-test
```

登录并设置特定集群和命名空间

```
ac login https://example.com --name prod --cluster=my-cluster --namespace=my-namespace
```

使用自定义 kubeconfig 文件登录

```
ac login https://example.com --name prod --kubeconfig=/path/to/kubeconfig
```

ac logout

结束当前 ACP 平台会话

示例用法

登出当前 ACP 平台会话

```
ac logout
```

登出指定会话

```
ac logout --session prod
```

登出所有会话

```
ac logout --all
```

ac logs

打印 pod 中容器的日志

示例用法

返回单容器 pod nginx 的快照日志

```
ac logs nginx
```

返回 pod nginx 的快照日志，前缀为源 pod 和容器名称

```
ac logs nginx --prefix
```

返回 pod nginx 的快照日志，限制输出为 500 字节

```
ac logs nginx --limit-bytes=500
```

返回 pod nginx 的快照日志，等待最多 20 秒以启动运行

```
ac logs nginx --pod-running-timeout=20s
```

返回多容器 pod nginx 的快照日志

```
ac logs nginx --all-containers=true
```

返回部署 nginx 中所有 pods 的快照日志

```
ac logs deployment/nginx --all-pods=true
```

返回标签 app=nginx 定义的所有 pods 中所有容器的快照日志

```
ac logs -l app=nginx --all-containers=true
```

返回标签 app=nginx 定义的所有 pods 的快照日志，限制并发日志请求为 10 个 pods

```
ac logs -l app=nginx --max-log-requests=10
```

返回 pod web-1 中先前终止的 ruby 容器日志快照

```
ac logs -p -c ruby web-1
```

开始流式输出 pod nginx 的日志，即使发生错误也继续

```
ac logs nginx -f --ignore-errors=true
```

开始流式输出 pod web-1 中 ruby 容器的日志

```
ac logs -f -c ruby web-1
```

开始流式输出标签 app=nginx 定义的所有 pods 中所有容器的日志

```
ac logs -f -l app=nginx --all-containers=true
```

仅显示 pod nginx 最近 20 行日志

```
ac logs --tail=20 nginx
```

显示 pod nginx 最近一小时内的所有日志

```
ac logs --since=1h nginx
```

显示 pod nginx 从 2024 年 8 月 30 日 06:00:00 UTC 开始带时间戳的日志

```
ac logs nginx --since-time=2024-08-30T06:00:00Z --timestamps=true
```

```
# 显示 kubelet 的日志, 跳过过期的服务证书校验
```

```
ac logs --insecure-skip-tls-verify-backend nginx
```

```
# 返回名为 hello 的 job 的第一个容器日志快照
```

```
ac logs job/hello
```

```
# 返回部署 nginx 中容器 nginx-1 的日志快照
```

```
ac logs deployment/nginx -c nginx-1
```

ac namespace

显示或切换当前命名空间上下文

示例用法

```
# 显示当前命名空间和上下文信息
```

```
ac namespace
```

```
# 切换到不同的命名空间
```

```
ac namespace my-namespace
```

```
# 切换到默认命名空间
```

```
ac namespace default
```

ac patch

更新资源字段

示例用法

```
# 使用战略合并补丁部分更新节点，补丁为 JSON 格式
ac patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# 使用战略合并补丁部分更新节点，补丁为 YAML 格式
ac patch node k8s-node-1 -p '$spec:\n unschedulable: true'

# 使用战略合并补丁部分更新通过类型和名称指定的节点，配置文件为 "node.json"
ac patch -f node.json -p '{"spec":{"unschedulable":true}}'

# 更新容器镜像；spec.containers[*].name 是合并键
ac patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-
hostname","image":"new image"}]}}'

# 使用 JSON 补丁更新容器镜像，数组使用位置索引
ac patch pod valid-pod --type='json' -p='[{"op": "replace", "path":
"/spec/containers/0/image", "value":"new image"}]}'

# 通过 'scale' 子资源使用合并补丁更新部署副本数
ac patch deployment nginx-deployment --subresource='scale' --type='merge' -p '{"spec":
{"replicas":2}}'
```

ac plugin

提供插件交互工具

示例用法

```
# 列出所有可用插件
ac plugin list

# 仅列出可用插件的二进制名称，不含路径
ac plugin list --name-only
```

ac plugin list

列出用户 PATH 中所有可见的插件可执行文件

示例用法

```
# 列出所有可用插件
ac plugin list

# 仅列出可用插件的二进制名称，不含路径
ac plugin list --name-only
```

ac port-forward

将一个或多个本地端口转发到 pod

示例用法

```
# 本地监听端口 5000 和 6000，转发到 pod 中的 5000 和 6000 端口
ac port-forward pod/mypod 5000 6000

# 本地监听端口 5000 和 6000，转发到由部署选择的 pod 中的 5000 和 6000 端口
ac port-forward deployment/mydeployment 5000 6000

# 本地监听端口 8443，转发到由服务选择的 pod 中名为 "https" 的服务端口的 targetPort
ac port-forward service/myservice 8443:https

# 本地监听端口 8888，转发到 pod 中的 5000 端口
ac port-forward pod/mypod 8888:5000

# 本地监听所有地址的端口 8888，转发到 pod 中的 5000 端口
ac port-forward --address 0.0.0.0 pod/mypod 8888:5000

# 本地监听 localhost 和指定 IP 的端口 8888，转发到 pod 中的 5000 端口
ac port-forward --address localhost,10.19.21.23 pod/mypod 8888:5000

# 本地监听随机端口，转发到 pod 中的 5000 端口
ac port-forward pod/mypod :5000
```

ac process

将模板处理为资源列表

示例用法

```
# 将 template.json 文件转换为资源列表并传递给 create
ac process -f template.json | ac apply -f -
```

```
# 本地处理文件而非联系服务器
ac process -f template.json -o yaml
```

```
# 处理模板时传递用户定义的标签
ac process -f template.json -l name=mytemplate
```

```
# 将存储的模板转换为资源列表
ac process foo
```

```
# 通过设置/覆盖参数值将存储的模板转换为资源列表
ac process foo -p PARM1=VALUE1 -p PARM2=VALUE2
```

```
# 将存储在不同命名空间的模板转换为资源列表
ac process cpaas-system//foo
```

```
# 将 template.json 转换为资源列表
cat template.json | ac process -f -
```

ac proxy

运行 Kubernetes API 服务器代理

示例用法


```
# 代理所有 Kubernetes API, 且不代理其他内容
ac proxy --api-prefix=/

# 代理部分 Kubernetes API 及一些静态文件
# 可通过 'curl localhost:8001/api/v1/pods' 获取 pods 信息
ac proxy --www=/my/files --www-prefix=/static/ --api-prefix=/api/

# 以不同根路径代理整个 Kubernetes API
# 可通过 'curl localhost:8001/custom/api/v1/pods' 获取 pods 信息
ac proxy --api-prefix=/custom/

# 在端口 8011 运行 Kubernetes API 代理, 静态内容来自 ./local/www/
ac proxy --port=8011 --www=./local/www/

# 在任意本地端口运行 Kubernetes API 代理
# 服务器选择的端口将输出到标准输出
ac proxy --port=0

# 运行 Kubernetes API 代理, 修改 API 前缀为 k8s-api
# 例如 pods API 可通过 localhost:8001/k8s-api/v1/pods/ 访问
ac proxy --api-prefix=/k8s-api
```

ac replace

通过文件名或标准输入替换资源

示例用法

```
# 使用 pod.json 中的数据替换 pod
ac replace -f ./pod.json

# 基于标准输入传入的 JSON 替换 pod
cat pod.json | ac replace -f -

# 更新单容器 pod 的镜像版本（标签）为 v4
ac get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | ac replace -f -

# 强制替换，先删除再重新创建资源
ac replace --force -f ./pod.json
```

ac rollout

管理资源的发布

示例用法

```
# 回滚到上一个部署版本
ac rollout undo deployment/abc

# 检查 daemonset 的发布状态
ac rollout status daemonset/foo

# 重启部署
ac rollout restart deployment/abc

# 重启带有标签 'app=nginx' 的部署
ac rollout restart deployment --selector=app=nginx
```

ac rollout history

查看发布历史

示例用法

```
# 查看部署的发布历史
ac rollout history deployment/abc

# 查看 daemonset 第 3 版的详细信息
ac rollout history daemonset/abc --revision=3
```

ac rollout pause

将指定资源标记为暂停

示例用法

```
# 将 nginx 部署标记为暂停
# 部署当前状态继续运行；暂停期间新更新不会生效
ac rollout pause deployment/nginx
```

ac rollout restart

重启资源

示例用法

```
# 重启 test-namespace 命名空间中的所有部署
ac rollout restart deployment -n test-namespace

# 重启部署
ac rollout restart deployment/nginx

# 重启 daemonset
ac rollout restart daemonset/abc

# 重启带有标签 app=nginx 的部署
ac rollout restart deployment --selector=app=nginx
```

ac rollout resume

恢复已暂停的资源

示例用法

```
# 恢复已暂停的部署
ac rollout resume deployment/nginx
```

ac rollout status

显示发布状态

示例用法

```
# 监视部署的发布状态
ac rollout status deployment/nginx
```

ac rollout undo

撤销先前的发布

示例用法

```
# 回滚到上一个部署版本
ac rollout undo deployment/abc

# 回滚到 daemonset 第 3 版
ac rollout undo daemonset/abc --to-revision=3

# 以 dry-run 模式回滚到上一个部署版本
ac rollout undo --dry-run=server deployment/abc
```

ac run

在集群上运行特定镜像

示例用法

```
# 启动 nginx pod
ac run nginx --image=nginx

# 启动 hazelcast pod, 并让容器暴露端口 5701
ac run hazelcast --image=hazelcast/hazelcast --port=5701

# 启动 hazelcast pod, 在容器中设置环境变量 "DNS_DOMAIN=cluster" 和
"POD_NAMESPACE=default"
ac run hazelcast --image=hazelcast/hazelcast --env="DNS_DOMAIN=cluster" --
env="POD_NAMESPACE=default"

# 启动 hazelcast pod, 在容器中设置标签 "app=hazelcast" 和 "env=prod"
ac run hazelcast --image=hazelcast/hazelcast --labels="app=hazelcast,env=prod"

# dry run ; 打印对应的 API 对象但不创建
ac run nginx --image=nginx --dry-run=client

# 启动 nginx pod, 使用默认命令, 但用自定义参数替代 (arg1 .. argN)
ac run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# 启动 nginx pod, 使用不同命令和自定义参数
ac run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>

# 启动 busybox pod, 保持前台运行, 退出时不重启
ac run -i -t busybox --image=busybox --restart=Never
```

ac scale

设置部署、ReplicaSet 或 ReplicationController 的新规模

示例用法

```
# 将名为 'foo' 的 replica set 扩缩到 3
ac scale --replicas=3 rs/foo

# 将通过类型和名称指定的资源（配置文件为 "foo.yaml"）扩缩到 3
ac scale --replicas=3 -f foo.yaml

# 如果名为 mysql 的部署当前规模为 2，则扩缩到 3
ac scale --current-replicas=2 --replicas=3 deployment/mysql

# 扩缩多个 replication controllers
ac scale --replicas=5 rc/example1 rc/example2 rc/example3

# 将名为 'web' 的 stateful set 扩缩到 3
ac scale --replicas=3 statefulset/web
```

ac set

设置对象的特定功能

ac set env

更新 pod 模板中的环境变量

示例用法

```

# 更新部署 'registry', 添加新环境变量
ac set env deployment/registry STORAGE_DIR=/local

# 列出部署 'sample-build' 中定义的环境变量
ac set env deployment/sample-build --list

# 列出所有 pods 中定义的环境变量
ac set env pods --all --list

# 输出修改后的部署 YAML, 不修改服务器上的对象
ac set env deployment/sample-build STORAGE_DIR=/data -o yaml

# 更新项目中所有 replication controllers 的所有容器, 设置 ENV=prod
ac set env rc --all ENV=prod

# 从 secret 导入环境变量
ac set env --from=secret/mysecret deployment/myapp

# 从 config map 导入环境变量, 添加前缀
ac set env --from=configmap/myconfigmap --prefix=MYSQL_ deployment/myapp

# 从 config map 导入指定键
ac set env --keys=my-example-key --from=configmap/myconfigmap deployment/myapp

# 从所有部署配置中删除容器 'c1' 的环境变量 ENV
ac set env deployments --all --containers="c1" ENV-

# 从磁盘上的部署定义中删除环境变量 ENV, 并更新服务器上的部署配置
ac set env -f deploy.json ENV-

# 将本地 shell 环境中的部分变量导入部署配置
env | grep RAILS_ | ac set env -e - deployment/registry

```

ac set image

更新 pod 模板的镜像

示例用法


```
# 设置部署中 nginx 容器镜像为 'nginx:1.9.1', busybox 容器镜像为 'busybox'
ac set image deployment/nginx busybox=busybox nginx=nginx:1.9.1

# 更新所有部署和 rc 中 nginx 容器镜像为 'nginx:1.9.1'
ac set image deployments,rc nginx=nginx:1.9.1 --all

# 更新 daemonset abc 中所有容器镜像为 'nginx:1.9.1'
ac set image daemonset abc *=nginx:1.9.1

# 打印本地文件更新 nginx 容器镜像的结果 (yaml 格式), 不访问服务器
ac set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml
```

ac set resources

更新带 pod 模板对象的资源请求/限制

示例用法

```
# 设置部署 nginx 容器的 cpu 限制为 "200m", 内存限制为 "512Mi"
ac set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# 设置 nginx 中所有容器的资源请求和限制
ac set resources deployment nginx --limits=cpu=200m,memory=512Mi --
requests=cpu=100m,memory=256Mi

# 移除 nginx 容器的资源请求
ac set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=0,memory=0

# 打印本地文件更新 nginx 容器限制的结果 (yaml 格式), 不访问服务器
ac set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --local -o yaml
```

ac set selector

设置资源的选择器

示例用法

```
# 在创建部署/服务对之前设置标签和选择器
ac create service clusterip my-svc --clusterip="None" -o yaml --dry-run=client | ac set
selector --local -f - 'environment=qa' -o yaml | ac create -f -
ac create deployment my-dep -o yaml --dry-run=client | ac label --local -f -
environment=qa -o yaml | ac create -f -
```

ac set serviceaccount

更新资源的服务账户

示例用法

```
# 设置部署 nginx-deployment 的服务账户为 serviceaccount1
ac set serviceaccount deployment nginx-deployment serviceaccount1

# 打印本地文件更新 nginx 部署服务账户的结果（YAML 格式），不访问 API 服务器
ac set sa -f nginx-deployment.yaml serviceaccount1 --local --dry-run=client -o yaml
```

ac set subject

更新角色绑定或集群角色绑定中的用户、组或服务账户

示例用法

```
# 更新集群角色绑定，指定服务账户 serviceaccount1
ac set subject clusterrolebinding admin --serviceaccount=namespace:serviceaccount1

# 更新角色绑定，指定用户 user1、user2 和组 group1
ac set subject rolebinding admin --user=user1 --user=user2 --group=group1

# 打印本地文件更新角色绑定主体的结果（YAML 格式），不访问服务器
ac create rolebinding admin --role=admin --user=admin -o yaml --dry-run=client | ac set
subject --local -f - --user=foo -o yaml
```

ac version

打印客户端和服务端版本信息

示例用法

```
# 打印客户端和服务端版本信息
ac version

# 仅打印客户端版本
ac version --client

# 以 JSON 格式打印版本信息
ac version -o json
```

ac wait

实验性功能：等待一个或多个资源满足特定条件

示例用法

等待 pod "busybox1" 包含类型为 "Ready" 的状态条件

```
ac wait --for=condition=Ready pod/busybox1
```

状态条件默认值为 true ; 可等待其他目标, 等号右侧比较时忽略大小写

```
ac wait --for=condition=Ready=false pod/busybox1
```

等待 pod "busybox1" 的状态 phase 为 "Running"

```
ac wait --for=jsonpath='{.status.phase}'=Running pod/busybox1
```

等待 pod "busybox1" 的 Ready 状态条件为 True

```
ac wait --for='jsonpath={.status.conditions[?(@.type=="Ready")].status}=True'
pod/busybox1
```

等待服务 "loadbalancer" 拥有 ingress

```
ac wait --for=jsonpath='{.status.loadBalancer.ingress}' service/loadbalancer
```

等待 secret "busybox1" 被创建, 超时 30 秒

```
ac create secret generic busybox1
```

```
ac wait --for=create secret/busybox1 --timeout=30s
```

删除 pod "busybox1" 后等待其被删除, 超时 60 秒

```
ac delete pod/busybox1
```

```
ac wait --for=delete pod/busybox1 --timeout=60s
```

AC CLI 管理员命令参考

本参考提供了 AC CLI 管理员命令的描述和示例命令。您必须具有 cluster-admin 或等效权限才能使用这些命令。

有关开发人员命令，请参见 AC CLI 开发人员命令参考。

运行 `ac adm -h` 可列出所有管理员命令，或运行 `ac <command> --help` 获取特定命令的详细信息。

目录

ac adm

示例用法

ac adm certificate

ac adm certificate approve

示例用法

ac adm certificate deny

示例用法

ac adm cordon

示例用法

ac adm drain

示例用法

ac adm new-project

示例用法

ac adm new-project-namespace

示例用法

ac adm policy

示例用法

```
ac adm policy add-cluster-role-to-user
```

示例用法

```
ac adm policy add-namespace-role-to-user
```

示例用法

```
ac adm policy add-project-role-to-user
```

示例用法

```
ac adm policy add-role-to-user
```

示例用法

```
ac adm taint
```

示例用法

```
ac adm uncordon
```

示例用法

ac adm

ACP 集群管理的管理工具

示例用法

```
# 排空节点以进行维护
```

```
ac adm drain NODE_NAME
```

```
# 标记节点为不可调度
```

```
ac adm cordon NODE_NAME
```

```
# new-project 用于在集群中创建项目
```

```
ac adm new-project --cluster CLUSTER_NAME
```

```
# 取消标记节点为不可调度（标记为可调度）
```

```
ac adm uncordon NODE_NAME
```

ac adm certificate

修改证书资源

ac adm certificate approve

批准证书签名请求

示例用法

```
# 批准 CSR 'csr-sqgzp'  
ac adm certificate approve csr-sqgzp
```

ac adm certificate deny

拒绝证书签名请求

示例用法

```
# 拒绝 CSR 'csr-sqgzp'  
ac adm certificate deny csr-sqgzp
```

ac adm cordon

将节点标记为不可调度

示例用法

```
# 将节点 "foo" 标记为不可调度
ac adm cordon foo
```

ac adm drain

排空节点以准备维护

示例用法

```
# 排空节点 "foo", 即使其上有不受 replication controller、replica set、job、daemon set 或
stateful set 管理的 Pod
ac adm drain foo --force

# 同上, 但如果存在不受上述控制器管理的 Pod 则中止, 并使用 15 分钟的宽限期
ac adm drain foo --grace-period=900
```

ac adm new-project

创建新项目

示例用法

```
# 在指定集群中创建项目
ac adm new-project my-project --cluster cluster1

# 在多个集群中创建项目
ac adm new-project my-project --cluster cluster1,cluster2
```

ac adm new-project-namespace

在项目中创建新命名空间

示例用法

```
# 在指定集群的项目中创建命名空间
ac adm new-project-namespace my-namespace --project my-project --cluster cluster1
```

ac adm policy

管理项目或命名空间的 RBAC 策略

示例用法

```
# 将用户分配到项目中的管理员角色
ac adm policy add-project-role-to-user project-admin-system alice --project my-project

# 将用户分配到项目中特定集群命名空间的命名空间角色
ac adm policy add-namespace-role-to-user namespace-developer-system alice --namespace my-namespace --project my-project --cluster business-1

# 给用户 alice 添加 kubernetes 集群角色 view
ac adm policy add-cluster-role-to-user view alice

# 给用户 alice 添加 kubernetes 角色 view（指定命名空间）
ac adm policy add-role-to-user view alice -n my-namespace
```

ac adm policy add-cluster-role-to-user

将 kubernetes 集群角色分配给当前上下文集群中的用户

示例用法

```
# 给用户 alice 添加 kubernetes 集群角色 view
ac adm policy add-cluster-role-to-user view alice
```

ac adm policy add-namespace-role-to-user

将平台角色分配给项目中特定集群命名空间的用户

示例用法

```
# 将 namespace-developer-system 角色分配给项目 my-project 中的用户 alice
ac adm policy add-namespace-role-to-user namespace-developer-system alice --namespace my-namespace --project my-project --cluster business-1
```

ac adm policy add-project-role-to-user

将平台角色分配给项目中的用户

示例用法

```
# 将 project-admin-system 角色分配给项目 my-project 中的用户 alice
ac adm policy add-project-role-to-user project-admin-system alice --project my-project
```

ac adm policy add-role-to-user

将 kubernetes 角色分配给当前上下文集群中的用户

示例用法

```
# 给用户 alice 添加 kubernetes 角色 view
ac adm policy add-role-to-user view alice -n my-namespace
```

ac adm taint

更新一个或多个节点上的污点

示例用法

```
# 给节点 'foo' 添加一个键为 'dedicated', 值为 'special-user', 效果为 'NoSchedule' 的污点
# 如果已有相同键和效果的污点, 则替换其值
ac adm taint nodes foo dedicated=special-user:NoSchedule

# 如果存在, 移除节点 'foo' 上键为 'dedicated', 效果为 'NoSchedule' 的污点
ac adm taint nodes foo dedicated:NoSchedule-

# 移除节点 'foo' 上所有键为 'dedicated' 的污点
ac adm taint nodes foo dedicated-

# 给带有标签 myLabel=X 的节点添加键为 'dedicated', 效果为 'PreferNoSchedule' 的污点
ac adm taint node -l myLabel=X dedicated=foo:PreferNoSchedule

# 给节点 'foo' 添加一个键为 'bar', 无值的污点, 效果为 'NoSchedule'
ac adm taint nodes foo bar:NoSchedule
```

ac adm uncordon

将节点标记为可调度

示例用法

```
# 将节点 "foo" 标记为可调度
ac adm uncordon foo
```