云容器实例 CCI 2.0

开发指南

文档版本 01

发布日期 2025-09-29

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 使用 ccictl

1.1 约束与限制

- 通过ccictl操作CCI资源暂不支持IAM 5.0(Landingzone)场景。
- 目前ccictl仅支持以下命令范围和资源范围:

  支持命令：create, set, explain, get, edit, delete, rollout, describe, logs, exec, apply, api-resources, api-versions, config, version, options

  支持资源及命令如下：

命令	namespace	network	deployment	pod	service	configmap	secret	hpa	pvc	pv
create	√	√	√	√	√	√	√	√	√	√
set	×	×	√	√	√	×	×	×	×	×
explain	√	√	√	√	√	√	√	√	√	√
get	√	√	√	√	√	√	√	√	√	√
edit	√	√	√	√	√	√	√	√	√	√
delete	√	√	√	√	√	√	√	√	√	√
rollout	×	×	√	×	×	×	×	×	×	×
describe	√	√	√	√	√	√	√	√	√	√
logs	×	×	×	√	×	×	×	×	×	×
exec	×	×	×	√	×	×	×	×	×	×
apply	√	√	√	√	√	√	√	√	√	√

- 进行命名空间级资源操作，如果未显示指定“--namespace”选项时，ccictl会默认指定“default”命名空间。新用户首次使用CCI，云服务不会自动创建“default”命名空间。

1.2 ccictl 配置指南

云容器实例支持使用定制的ccictl来创建负载等资源。

下载 ccictl

请在**表1**下载对应版本的ccictl。

表 1-1 ccictl 下载地址

操作系统	下载地址
Linux AMD 64位	**ccictl_linux-amd64** **ccictl_linux-amd64_sha256**
Mac AMD 64位	**ccictl_mac-amd64** **ccictl_mac-amd64_sha256**
Mac ARM 64位	**ccictl_mac-arm64** **ccictl_mac-arm64_sha256**

安装并设置 ccictl

以下操作以Linux环境为例。

步骤1 将**下载ccictl**中下载的ccictl赋予可执行权限，并放到PATH目录下。

```
#chmod +x ./ccictl
#mv ./ccictl $PATH
```

其中，*$PATH*为PATH路径（如/usr/local/bin），请替换为实际的值。

您还可以通过如下命令查看ccictl的版本，如下所示。

```
#ccictl version
#Client Version: version.Info{GitVersion:"v0.0.2",
GitCommit:"996a2efcd852473dde345ae2931833342cab1d96", BuildDate:"2025-03-24T00:32:05Z",
GoVersion:"go1.19.6", Compiler:"gc", Platform:"linux/amd64"}
```

步骤2 配置IAM认证信息并持久化到本地。

1. 指定CCI集群：

   ```
   ccictl config set-cluster <$context-cluster-name> --server=https://<$cci-endpoint>
   ```

   - $context-cluster-name：集群名称，要用户可自定义。
   - $cci-endpoint：CCI的Endpoint，不同区域的终端节点不同，详情请参考**获取云容器实例Endpoint**。

   示例：

   ```
   #ccictl config set-cluster cci-cluster --server=https://cci.cn-north-4.myhuaweicloud.com
   #Cluster "cci-cluster" set.
   ```

2. 设置ccictl使用的用户凭证：

```
ccictl config set-credentials <$user> --auth-provider=iam --auth-provider-arg=iam-endpoint=https://<$iam-endpoint> --auth-provider-arg=cache="true" --auth-provider-arg=project-name=<$project-name> --auth-provider-arg=user-name=<$user-name> --auth-provider-arg=domain-name=<$domain-name> --auth-provider-arg=password=<$password>
```

或

```
ccictl config set-credentials <$user> --auth-provider=iam --auth-provider-arg=iam-endpoint=https://<$iam-endpoint> --auth-provider-arg=cache="true" --auth-provider-arg=project-name=<$project-name> --auth-provider-arg=ak=<$ak> --auth-provider-arg=sk=<$sk>
```

表 1-2 用户名/密码配置

Command Flag	Description
domain-name	租户名，即账号名。
user-name	子用户名，即IAM用户名。
password	用户或子用户密码。

表 1-3 AK/SK 配置

Command Flag	Description
ak	ak、sk的获取方法请参见**获取AK/SK**，ak为文件中Access Key部分，sk为文件中Secret Key部分。
sk	

表 1-4 公共配置

Command Flag	Description
context-user-name	用户名，即配置的上下文用户名，用户可自定义。
iam-endpoint	IAM的Endpoint，必须配置，详情请参见**地区和终端节点**。
project-name	项目名称，详情请参见**从控制台获取项目名称**。
cache	是否开启将IAM Token缓存到本地，提高访问性能，默认为true。 **注意** 在非安全环境，建议关闭此选项。

示例：

```
#ccictl config set-credentials cci-user --auth-provider=iam --auth-provider-arg=iam-endpoint=https://{iam-endpoint} --auth-provider-arg=cache="true" --auth-provider-arg=project-name={region} --auth-provider-arg=ak=ak****** --auth-provider-arg=sk=sk******
#User "cci-user" set.
```

3. 设置和使用ccictl的上下文：

```
ccictl config set-context <$context-name> --cluster=<$context-cluster-name> --user=<$context-user-name>
```

- $context-name：上下文名称，用户可自定义。
- $context-cluster-name：前文配置的集群名称。
- $context-user-name：前文配置的用户名。

示例：

```
#ccictl config set-context cci-context1 --cluster=cci-cluster --user=cci-user
#Context "cci-context1" created.
```

使用ccictl的上下文。

```
ccictl config use-context  <$context-name>
```

示例：

```
#ccictl config use-context cci-context1
#Switched to context "cci-context1".
```

步骤3 配置完成后，即可通过ccictl命令操作云容器实例的相关资源。

例如，执行ccictl get namespace，查看namespace资源。

```
#ccictl get namespace
#No resources found.
```

----结束

非安全环境配置 ccictl

步骤1 参照上述**安装并设置ccictl**。

步骤2 编辑config文件，删除敏感信息参数。

Linux系统，config文件默认位于$HOME/.kubev2/config。

表 1-5 删除敏感信息参数

Command Flag	Environment Value	Description
--domain-name	DOMAIN_NAME	租户名
--user-name	USER_NAME	子用户名
--password	PASSWORD	用户密码
--ak	ACCESS_KEY_ID	Access Key
--sk	SECRET_ACCESS_KEY	Secret Key
--cache	CREDENTIAL_CACHE	是否开启缓存Token

步骤3 配置删除参数相应的环境变量来使用ccictl，以AK/SK为例。

```
#export ACCESS_KEY_ID=xxxxxxx
#export SECRET_ACCESS_KEY=xxxxxxx
#export CREDENTIAL_CACHE=false
#ccictl get ns
```

执行上述命令后，提示如下类似信息：

```
#No resources found.
```

----结束

获取 AK/SK

AK(Access Key ID)：访问密钥ID。与私有访问密钥关联的唯一标识符；访问密钥ID和私有访问密钥一起使用，对请求进行加密签名。

SK(Secret Access Key)：与访问密钥ID结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

步骤1 登录管理控制台。

步骤2 单击用户名，在下拉列表中单击“我的凭证”。

步骤3 在“我的凭证”页面，单击“访问密钥”页签。

步骤4 单击“新增访问密钥”，输入验证码。

步骤5 单击“确定”，生成并下载访问密钥。

说明

为防止访问密钥泄露，建议您将其保存到安全的位置。

----结束

获取云容器实例 Endpoint

终端节点（Endpoint）即调用API的**请求地址**，不同服务不同区域的终端节点不同，您可以从**地区和终端节点**查询服务的终端节点。

请您根据业务需要选择对应区域的终端节点。

表 1-6 地区和终端节点

区域名称	区域	终端节点（Endpoint）
土耳其-伊斯坦布尔	tr-west-1	cci.tr-west-1.myhuaweicloud.com
非洲-约翰内斯堡	af-south-1	cci.af-south-1.myhuaweicloud.com
亚太-新加坡	ap-southeast-3	cci.ap-southeast-3.myhuaweicloud.com
中东-利雅得	me-east-1	cci.me-east-1.myhuaweicloud.com
拉美-墨西哥城二	la-north-2	cci.la-north-2.myhuaweicloud.com
拉美-圣保罗一	sa-brazil-1	cci.sa-brazil-1.myhuaweicloud.com

区域名称	区域	终端节点（Endpoint）
拉美-圣地亚哥	la-south-2	cci.la-south-2.myhuaweicloud.com

从控制台获取项目名称

从控制台获取项目名称的步骤如下：

1. 登录管理控制台。
2. 鼠标悬停在右上角的用户名，选择下拉列表中的“我的凭证”。
   在“API凭证”页面的项目列表中查看项目。

1.3 命令行工具(ccictl)

1.3.1 ccictl 概述

ccictl 介绍

ccictl是类似kubectl的，使用API与CCI 2.0服务进行通信的CLI版本命令行工具。其可胜任管理CCI 2.0集群的多种任务。

针对配置信息，ccictl 在 $HOME/.kubev2 目录中查找一个名为 config 的配置文件。您可以通过设置 CLICONFIG 环境变量或设置 --cliconfig 参数来指定其他config文件。

本章节主要介绍如何使用ccictl在CCI 2.0中声明式管理应用，本章节还涵盖一些其他的ccictl功能。

⚠ 注意

CCI 2.0服务侧提供的所有资源所属组和版本(group/version)基本上属于cci/v2，ccictl工具创建，查询，删除等操作的资源的group/version也基本上属于cci/v2，所以直接使用group/version为v1或apps/v1的资源文件作为ccictl的入参，命令执行会失败。

命令分类

大多数 ccictl 命令通常可以分为以下几类：

表 1-7 ccictl 命名分类

类型	用途	描述
声明式资源管理	部署和运维（如 GitOps）	使用资源管理声明式管理CCI 2.0工作负载。
命令式资源管理	调试	使用命令行参数和标志来管理CCI 2.0工作负载。

类型	用途	描述
打印工作负载状态	调试	打印有关工作负载的信息。
与容器交互	调试	执行、挂接、复制、日志。

声明式应用管理

管理资源的首选方法是配合 ccictl apply 命令一起使用名为资源的声明式文件。 此命令读取本地（或远程）文件结构，并修改集群状态以反映声明的意图。

说明

Apply
Apply 是在CCI 2.0集群中管理资源的首选机制。

打印工作负载状态

ccictl 支持通过提供以下命令进行调试：

- 打印 Container 日志
- 打印事件
- 执行到 Container

语法

使用以下语法从终端窗口运行ccictl命令：

```
ccictl [command] [TYPE] [NAME] [flags]
```

其中 command、TYPE、NAME 和 flags 分别是：

- command：指定要对一个或多个资源执行的操作，例如 create、get、describe、delete。
- TYPE：指定资源类型。资源类型不区分大小写， 可以指定单数、复数或缩写形式。例如，以下命令输出相同的结果：

```
ccictl get pod pod1
ccictl get pods pod1
ccictl get po pod1
```

- NAME：指定资源的名称。名称区分大小写。 如果省略名称，则显示所有资源的详细信息。例如：ccictl get pods。

  在对多个资源执行操作时，您可以按类型和名称指定每个资源，或指定一个或多个文件：
- 要按类型和名称指定资源：
- 要对所有类型相同的资源进行分组，请执行以下操作：TYPE1 name1 name2 name<#>。

  示例：ccictl get pod example-pod1 example-pod2
- 分别指定多个资源类型：TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>。

示例：ccictl get pod/example-pod1 deployment/example-deploy1

- 用一个或多个文件指定资源：-f file1 -f file2 -f file<#>
- 建议使用 YAML 而不是 JSON， 因为 YAML 对用户更友好, 特别是对于配置文件。

  示例：ccictl get -f ./pod.yaml
- flags： 指定可选的参数。例如，可以使用 -s 或 --server 参数指定 API 服务器的地址和端口。

⚠ 注意

从命令行指定的参数会覆盖默认值和任何相应的环境变量。

如果您需要帮助，在终端窗口中运行 ccictl help。

操作

下表包含所有ccictl操作的简短描述和普通语法：

表 1-8 描述和语法

操作	语法	描述
api-resources	ccictl api-resources [flags]	列出可用的 API 资源。
api-versions	ccictl api-versions [flags]	列出可用的 API 版本。
apply	ccictl apply -f FILENAME [flags]	从文件或 stdin 对资源应用配置更改。
config	ccictl config SUBCOMMAND [flags]	修改 config 文件。有关详细信息，请参阅各个子命令。
create	ccictl create -f FILENAME [flags]	从文件或 stdin 创建一个或多个资源。
delete	ccictl delete (-f FILENAME | TYPE [NAME | /NAME | -l label | --all]) [flags]	基于文件、标准输入或通过指定标签选择器、名称、资源选择器或资源本身，删除资源。
describe	ccictl describe (-f FILENAME | TYPE [NAME_PREFIX | /NAME | -l label]) [flags]	显示一个或多个资源的详细状态。
edit	ccictl edit (-f FILENAME | TYPE NAME | TYPE/NAME) [flags]	使用默认编辑器编辑和更新服务器上一个或多个资源的定义。

操作	语法	描述
exec	ccictl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]	对 Pod 中的容器执行命令。
explain	ccictl explain TYPE [--recursive=false] [flags]	获取多种资源的文档。例如 Pod、Deployment、Service 等。
get	ccictl get (-f FILENAME | TYPE [NAME | /NAME | -l label]) [--watch] [--sort-by=FIELD] [[-o | --output]=OUTPUT_FORMAT] [flags]	列出一个或多个资源。
logs	ccictl logs POD [-c CONTAINER] [--follow] [flags]	打印 Pod 中容器的日志。
rollout	ccictl rollout SUBCOMMAND [options]	管理资源的上线。有效的资源类型：Deployment
set	ccictl set SUBCOMMAND [options]	配置应用资源。
version	ccictl version [flags]	显示运行在客户端的 ccictl 版本。

资源类型

表 1-9 资源类型

资源名	缩写名	API版本	按命名空间	资源类型
configmaps	cm	cci/v2	true	ConfigMap
deployments	deploy	cci/v2	true	Deployment
horizontalpodautoscalers	hpa	cci/v2	true	HorizontalPodAutoscaler
namespaces	ns	cci/v2	false	Namespace
persistentvolumeclaims	pvc	cci/v2	true	PersistentVolumeClaim
persistentvolumes	pv	cci/v2	false	PersistentVolume
pods	po	cci/v2	true	Pod

资源名	缩写名	API版本	按命名空间	资源类型
replicasets	rs	cci/v2	true	ReplicaSet
secrets	-	cci/v2	true	Secret
services	svc	cci/v2	true	Service
storageclasses	-	cci/v2	false	StorageClass
networks	-	yangtse/v2	true	Network

⚠ 注意

资源类型仅表明CCI 2.0支持的所有资源类型，不保证上表所有资源对象都提供GET, POST, PUT, DELETE, PATCH所有API，如replicasets资源对象仅提供命名空间级别的查询API。不同局点的CCI 2.0服务也存在API偏差。

当遇到存在争议的API或者资源类型时，请优先咨询CCI服务接口人，一切解释权归CCI服务所有。

输出选项

输出格式

所有 ccictl 命令的默认输出格式都是人类可读的纯文本格式。要以特定格式在终端窗口输出详细信息， 可以将 -o 或 --output 参数添加到受支持的 ccictl 命令中。

语法

```
ccictl [command] [TYPE] [NAME] -o <output_format>
```

支持的输出格式如下：

表 1-10

输出格式	描述
-o custom-columns=<spec>	使用逗号分隔的自定义列列表打印表。
-o custom-columns-file=<filename>	使用 <filename> 文件中的自定义列模板打印表。
-o json	输出 JSON 格式的 API 对象
-o jsonpath=<template>	打印 JSONPath 表达式定义的字段
-o jsonpath-file=<filename>	打印 <filename> 文件中 JSONPath 表达式定义的字段。
-o name	仅打印资源名称而不打印任何其他内容。
-o wide	以纯文本格式输出，包含所有附加信息。

输出格式	描述
-o yaml	输出 YAML 格式的 API 对象。

示例

在此示例中，以下命令将单个 Pod 的详细信息输出为 YAML 格式的对象：

```
ccictl get pod web-pod-13je7 -o yaml
```

自定义列

要定义自定义列并仅将所需的详细信息输出到表中，可以使用 custom-columns 选项。 您可以选择内联定义自定义列或使用模板文件：-o custom-columns=<spec> 或 -o custom-columns-file=<filename>。

示例：

内联：

```
ccictl get pods <pod-name> -o custom-columns=NAME:.metadata.name,RSRC:.metadata.resourceVersion
```

模板文件：

```
ccictl get pods <pod-name> -o custom-columns-file=template.txt
```

其中，template.txt 文件包含：

```
NAME         RSRC
metadata.name metadata.resourceVersion
```

运行这两个命令之一的结果类似于：

```
NAME          RSRC
example-name   610995
```

排序列表对象

要将对象排序后输出到终端窗口，可以将 --sort-by 参数添加到支持的 ccictl 命令。 通过使用 --sort-by 参数指定任何数字或字符串字段来对对象进行排序。 要指定字段，请使用 **JSONPath** 表达式。

语法

```
ccictl [command] [TYPE] [NAME] --sort-by=<jsonpath_exp>
```

示例

要打印按名称排序的 Pod 列表，请运行：

```
ccictl get pods --sort-by=.metadata.name
```

示例：常见操作

使用以下示例集来帮助您熟悉运行常用 ccictl 操作：

ccictl apply - 以文件或标准输入为准应用或更新资源。

```
# 使用 example-service.yaml 中的定义创建 Service。
ccictl apply -f example-service.yaml
```

```
# 使用 example-deployment.yaml 中的定义创建 deployment。
ccictl apply -f example-deployment.yaml

# 使用 <directory> 路径下的任意 .yaml、.yml 或 .json 文件 创建对象。
ccictl apply -f <directory>
```

ccictl get - 列出一个或多个资源。

```
# 以纯文本输出格式列出所有 Pod。
ccictl get pods

# 以纯文本输出格式列出所有 Pod，并包含附加信息。
ccictl get pods -o wide

# 以纯文本输出格式列出所有Pod和 Service。
ccictl get po,services

# 列出所有状态为 Pending 的 Pod
ccictl get pods --field-selector=status.phase=Pending
```

ccictl describe - 显示一个或多个资源的详细状态，默认情况下包括未初始化的资源。

```
# 显示名为 <service-name> 的 Service 的详细信息。
ccictl describe svc <service-name>

# 显示名为 <pod-name> 的 Pod 的详细信息。
ccictl describe pods/<pod-name>

# 显示由名为 <deploy-name> 的工作负载管理的所有 Pod 的详细信息。
ccictl describe pods <deploy-name>

# 描述所有的 Pod
ccictl describe pods
```

说明

ccictl get 命令通常用于检索同一资源类别的一个或多个资源。 它具有丰富的参数，允许您使用 -o 或 --output 参数自定义输出格式。 您可以指定 -w 或 --watch 参数以开始监测特定对象的更新。 ccictl describe 命令更侧重于描述指定资源的许多相关方面。它可以调用对 API 服务器 的多个 API 调用来为用户构建视图。 例如，该 ccictl describe pod 命令不仅检索有关Pod的信息，还检索与其关联生成的事件等。

ccictl delete - 基于文件、标准输入或通过指定标签选择器、名称、资源选择器或资源来删除资源。

```
# 使用 pod.yaml 文件中指定的类型和名称删除 Pod。
ccictl delete -f pod.yaml

# 删除所有带有 '<label-key>=<label-value>' 标签的 Pod 和 Service。
ccictl delete pods,services -l <label-key>=<label-value>

# 删除所有 Pod，包括未初始化的 Pod。
ccictl delete pods --all
```

ccictl exec - 对 Pod 中的容器执行命令。

```
# 从 Pod <pod-name> 中获取运行 'date' 的输出。默认情况下，输出来自第一个容器。
ccictl exec <pod-name> -- date

# 运行输出 'date' 获取在 Pod <pod-name> 中容器 <container-name> 的输出。
ccictl exec <pod-name> -c <container-name> -- date

# 获取一个交互 TTY 并在 Pod  <pod-name> 中运行 /bin/bash。默认情况下，输出来自第一个容器。
ccictl exec -ti <pod-name> -- /bin/bash
```

ccictl logs - 打印 Pod 中容器的日志。

```
# 返回 Pod <pod-name> 的日志快照。
ccictl logs <pod-name>
```



```
# 从 Pod <pod-name> 开始流式传输日志。这类似于 'tail -f' Linux 命令。
ccictl logs -f <pod-name>
```

JSONPath 支持

ccictl 工具支持 JSONPath 模板作为输出格式。

JSONPath 模板由大括号 { 和 } 包起来的 JSONPath 表达式组成。 ccictl 使用 JSONPath 表达式来过滤 JSON 对象中的特定字段并格式化输出。 除了原始的 JSONPath 模板语法，以下函数和语法也是有效的:

- 使用双引号将 JSONPath 表达式内的文本引起来。
- 使用 range，end 运算符来迭代列表。
- 使用负片索引后退列表。负索引不会“环绕”列表， 并且只要 \((- index + listLength) \ge 0 \) 就有效。

说明

- $ 运算符是可选的，因为默认情况下表达式总是从根对象开始。
- 结果对象将作为其 String() 函数输出。

JSONPath 中的函数

给定 JSON 输入:

```
{
  "kind": "List",
  "items":[
    {
      "kind":"None",
      "metadata":{
        "name":"127.0.0.1",
        "labels":{
          "kubernetes.io/hostname":"127.0.0.1"
        }
      },
      "status":{
        "capacity":{"cpu":"4"},
        "addresses":[{"type": "LegacyHostIP", "address":"127.0.0.1"}]
      }
    },
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.2"},
      "status":{
        "capacity":{"cpu":"8"},
        "addresses":[
          {"type": "LegacyHostIP", "address":"127.0.0.2"},
          {"type": "another", "address":"127.0.0.3"}
        ]
      }
    }
  ],
  "users":[
    {
      "name": "myself",
      "user": {}
    },
    {
      "name": "e2e",
      "user": {"username": "admin", "password": "secret"}
    }
  ]
}
```

表 1-11 函数说明

函数	描述	示例	结果
text	纯文本	kind is {.kind}	kind is List
@	当前对象	{@}	与输入相同
. 或 []	子运算符	{.kind}、{['kind']} 或 {['name \.type']}	List
..	递归下降	{..name}	127.0.0.1 127.0.0.2 myself e2e
*	通配符。获取所有对象	{.items[*].metadata.name}	[127.0.0.1 127.0.0.2]
[start:end:step]	下标运算符	{.users[0].name}	myself
[,]	并集运算符	{.items[*]['metadata.name', 'status.capacity']}	127.0.0.1 127.0.0.2 map[cpu:4] map[cpu:8]
?()	过滤	{.users[?(@.name=="e2e")].user.password}	secret
range，end	迭代列表	{range .items[*]}[{.metadata.name}, {.status.capacity}] {end}	[127.0.0.1, map[cpu:4]] [127.0.0.2, map[cpu:8]]
''	引用解释执行字符串	{range .items[*]}{.metadata.name}{'\t'}{end}	127.0.0.1 127.0.0.2
\	转义终止符	{.items[0].metadata.labels.kubernetes\.io/hostname}	127.0.0.1

通过 ccictl 使用 JSONPath 表达式

使用 ccictl 和 JSONPath 表达式的示例：

```
ccictl get pods -o json
ccictl get pods -o=jsonpath='{@}'
ccictl get pods -o=jsonpath='{.items[0]}'
ccictl get pods -o=jsonpath='{.items[0].metadata.name}'
ccictl get pods -o=jsonpath="{.items[*]['metadata.name', 'status.capacity']}"
ccictl get pods -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.startTime}{"\n"}{end}'
ccictl get pods -o=jsonpath='{.items[0].metadata.labels.kubernetes\.io/hostname}'
```

JSONPath 中的正则表达式

不支持 JSONPath 正则表达式。如需使用正则表达式进行匹配操作，您可以使用如 jq 之类的工具。

```
# ccictl 的 JSONpath 输出不支持正则表达式
# 下面的命令不会生效
ccictl get pods -o jsonpath='{.items[?(@.metadata.name=~/^test$/)].metadata.name}'

# 下面的命令可以获得所需的结果
ccictl get pods -o json | jq -r '.items[] | select(.metadata.name | test("test-")).metadata.name'
```

1.3.2 ccictl

操作背景

ccictl 用于控制CCI 2.0集群管理器。

```
ccictl [flag]
```

选项

```
--cache-dir string     默认值："$HOME/.kubev2/cache"
```

默认缓存目录。

```
--cliconfig string
```

CLI 请求要使用的 cliconfig 文件的路径。

```
--cluster string
```

要使用的 cliconfig 中的集群名称。

```
--context string
```

要使用的 cliconfig 上下文的名称。

```
--insecure-skip-tls-verify
```

如果为 true，则不检查服务器证书的有效性。

```
-n, --namespace string
```

如果存在，则是此 CLI 请求的命名空间范围。

```
--password string
```

对 API 服务器进行基本身份验证所用的密码。

```
--request-timeout string     默认值："0"
```

在放弃某个服务器请求之前等待的时长。非零值应包含相应的时间单位（例如 1s、2m、3h）。 值为零表示请求不会超时。

```
-s, --server string
```

API 服务器的地址和端口。

```
--token string
```

向 API 服务器进行身份验证的持有者令牌。

```
--user string
```

要使用的 cliconfig 用户的名称。

```
--username string
```

对 API 服务器进行基本身份验证时所用的用户名。

```
-v, --v int
```

指定本次命令的日志等级，数字越大打印的日志细节越多。

1.3.3 ccictl create

操作背景

基于文件或标准输入创建一个资源。

接受 JSON 和 YAML 格式。

```
ccictl create -f FILENAME
```

示例

```
# 使用 pod.json 中的数据创建一个 Pod
ccictl create -f ./pod.json

# 基于传入到标准输入的 JSON 创建一个 Pod
cat pod.json | ccictl create -f -

# 以 JSON 编辑 registry.yaml 中的数据，然后使用已编辑的数据来创建资源
ccictl create -f registry.yaml --edit -o json
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--edit
```

在创建之前编辑 API 资源。

```
-f, --filename strings
```

用于创建资源的文件名、目录或文件 URL。

```
-h, --help
```

create 操作的帮助命令。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--raw string
```

用于向服务器发送 POST 请求的原始 URI。使用 cliconfig 文件中指定的传输方式。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--save-config
```

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

--validate string[="strict"]　默认值：strict"

必须是以下选项之一：strict（或 true）、warn、ignore（或 false）。 "true" 或 "strict" 将使用模式定义来验证输入，如果无效，则请求失败。

"false" 或 "ignore" 将不会执行任何模式定义检查，而是静默删除所有未知或重复的字段。

--windows-line-endings

仅在 --edit=true 时相关。默认为您所用平台原生的行结尾格式。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.1 ccictl create namespace

操作背景

用指定的名称创建命名空间。

```
ccictl create namespace NAME [options]
```

示例

```
# 新建一个名为 my-namespace 的命名空间
ccictl create namespace my-namespace
```

选项

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

-h, --help

namespace 操作的帮助命令。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--save-config

如果为 true，则当前对象的配置将被保存在其注解中。否则，注解将保持不变。 当您希望后续对此对象执行 `ccictl apply` 操作时，此标志很有用。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.2 ccictl create secret docker-registry

操作背景

新建一个 Docker 仓库所用的 Secret。

Dockercfg Secret 用于向 Docker 仓库进行身份认证。

当使用 Docker 命令行推送镜像时，您可以通过运行以下命令向给定的仓库进行身份认证：

```
docker login DOCKER_REGISTRY_SERVER --username=DOCKER_USER --password=DOCKER_PASSWORD --email=DOCKER_EMAIL
```

这一命令会生成一个 ~/.dockercfg 文件，后续的 docker push 和 docker pull 命令将使用该文件向 Docker 仓库做身份认证。电子邮件地址是可选的。

在创建应用时，您可能有一个 Docker 仓库要求进行身份认证。为了让节点代表您拉取镜像，这些节点必须有凭据。 您可以通过创建一个 dockercfg Secret 并将其附加到您的服务账户来提供这种凭据信息。

```
ccictl create secret docker-registry NAME --docker-username=user --docker-password=password --docker-email=email [--docker-server=string] [--from-file=[key=]source]
```

示例

```
# 如果没有 .dockercfg 文件，可以直接创建一个 dockercfg Secret
ccictl create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL

# 基于 ~/.docker/config.json 新建一个名为 my-secret 的 Secret
ccictl create secret docker-registry my-secret --from-file=path/to/.docker/config.json
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--append-hash
```

将 Secret 的哈希值追加到它的名称上。

```
--docker-email string
```

用于访问 Docker 仓库的电子邮件。

```
--docker-password string
```

用于向 Docker 仓库做身份认证的密码。

```
--docker-server string    默认值："https://index.docker.io/v1/"
```

Docker 仓库所在的服务器地址。

```
--docker-username string
```

Docker 仓库身份认证所用的用户名。

```
--from-file strings
```

密钥文件可以通过其文件路径指定，这种情况将为它们分配一个默认名称 .dockerconfigjson； 也可以选择指定名称和文件路径，这种情况将使用给定的名称。 指定一个目录将遍历目录中所有已命名的且是有效 Secret 密钥的文件。 对于此命令，密钥应始终为 .dockerconfigjson。

```
-h, --help
```

docker-registry 操作的帮助命令。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--save-config
```

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.3 ccictl create secret generic

操作背景

基于文件、目录或指定的文字值创建 Secret。

单个 Secret 可以包含一个或多个键值对。

当基于文件创建 Secret 时，键将默认为文件的基本名称，值将默认为文件内容。 如果基本名称是无效的键，或者您希望选择自己的键，您可以指定一个替代键。

当基于目录创建 Secret 时，目录中每个基本名称为有效键的文件都将被打包到 Secret 中。 除常规文件外的所有目录条目（ 例如子目录、符号链接、设备、管道等 ）都将被忽略。

```
ccictl create secret generic NAME [--type=string] [--from-file=[key=]source] [--from-literal=key1=value1]
```

示例

```
# 新建一个名为 my-secret 的 Secret，其键为文件夹 bar 中的每个文件
ccictl create secret generic my-secret --from-file=path/to/bar

# 使用指定的键而不是磁盘上的文件名来新建一个名为 my-secret 的 Secret
ccictl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-file=ssh-publickey=path/to/id_rsa.pub

# 使用 key1=supersecret 和 key2=topsecret 新建一个名为 my-secret 的 Secret
ccictl create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

# 组合使用文件和文字值新建一个名为 my-secret 的 Secret
ccictl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-literal=passphrase=topsecret

# 使用 env 文件新建一个名为 my-secret 的 Secret
ccictl create secret generic my-secret --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

选项

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

--append-hash

将 Secret 的哈希值追加到它的名称上。

--from-env-file strings

指定文件路径以读取 key=val 对的行来创建一个 Secret。

--from-file strings

键可以通过其文件路径被指定，这种情况将为它们分配一个默认名称； 键也可以选择通过某个名称和文件路径被指定，这种情况将使用给定的名称。 指定一个目录将遍历目录中所有已命名的且是有效 Secret 键的文件。

--from-literal strings

指定键和文字值以插入到 Secret 中（ 例如 mykey=somevalue ）。

-h, --help

generic 操作的帮助命令。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--save-config

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

--type string

要创建的 Secret 的类别。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.4 ccictl create secret tls

操作背景

使用给定的公钥/私钥对创建 TLS Secret。

事先公钥/私钥对必须存在。公钥证书必须是以 .PEM 编码的，并且与给定的私钥匹配。

```
ccictl create secret tls NAME --cert=path/to/cert/file --key=path/to/key/file
```

示例

```
# 使用给定的密钥对新建一个名为 tls-secret 的 TLS Secret
ccictl create secret tls tls-secret --cert=path/to/tls.crt --key=path/to/tls.key
```

选项

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

--append-hash

将 Secret 的哈希值追加到它的名称上。

--cert string

PEM 编码的公钥证书的路径。

-h, --help

tls 操作的帮助命令。

--key string

与给定证书关联的私钥的路径。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--save-config

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.5 ccictl create configmap

操作背景

基于文件、目录或指定的文字值创建 ConfigMap。

一个 ConfigMap 可以包含一个或多个键/值对。

当您基于文件创建 ConfigMap 时，键默认为文件的基本名称，值默认为文件内容。如果基本名称是无效的键，您可以指定一个替代键。

当基于目录创建 ConfigMap 时，目录中每个基本名称是有效键的文件都会被打包到 ConfigMap 中。 除常规文件之外的所有目录条目都会被忽略（例如子目录、符号链接、设备、管道等）。

```
ccictl create configmap NAME [--from-file=[key=]source] [--from-literal=key1=value1]
```

示例

```
# 基于 bar 文件夹新建一个名为 my-config 的 ConfigMap
ccictl create configmap my-config --from-file=path/to/bar

# 新建一个名为 my-config 的 ConfigMap，使用指定的键而不是磁盘上的文件基本名称
ccictl create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-file=key2=/path/to/bar/
file2.txt

# 新建一个名为 my-config 的 ConfigMap，包含 key1=config1 和 key2=config2
ccictl create configmap my-config --from-literal=key1=config1 --from-literal=key2=config2

# 从文件中的 key=value 对新建一个名为 my-config 的 ConfigMap
ccictl create configmap my-config --from-file=path/to/bar

# 从 env 文件新建一个名为 my-config 的 ConfigMap
ccictl create configmap my-config --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

选项

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

--append-hash

将 Configmap 的哈希值追加到它的名称上。

--from-env-file strings

指定文件路径以读取 key=val 对的行来创建一个 Configmap 。

--from-file strings

键文件可以使用其文件路径来指定，在这种情况下，文件的基本名称将用作 ConfigMap 的键。 另外，键文件也可以选择使用键和文件路径来指定，在这种情况下，将使用指定的键。 指定一个目录将遍历此目录中所有被命名的文件（其基本名称为有效的 ConfigMap 键）。

--from-literal strings

指定键和文字值以插入到 ConfigMap 中（例如 mykey=somevalue）。

-h, --help

configmap 操作的帮助命令。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--save-config

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.6 ccictl create service loadbalancer

操作背景

创建指定名称的 LoadBalancer 类型 Service。

```
ccictl create service loadbalancer NAME [--tcp=port:targetPort] [--elb-id:targetElbID]
```

示例

```
# 新建名为 my-lbs 的 LoadBalancer 类型 Service
ccictl create service loadbalancer my-lbs --tcp=5678:8080 --elb-id=xxx
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-h, --help
```

loadbalancer 操作的帮助命令。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--save-config
```

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

```
--tcp strings
```

端口对可以指定为 "<端口>:<目标端口>"。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

```
--elb-id string
```

elb服务实例id

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.7 ccictl create service externalname

操作背景

创建指定名称的 ExternalName Service。

ExternalName Service 引用外部 DNS 地址，而不仅仅是 Pod， 这类 Service 允许应用作者引用平台外、其他集群或本地存在的服务。

```
ccictl create service externalname NAME --external-name external.name
```

示例

```
# 新建一个名为 my-ns 的 ExternalName Service
ccictl create service externalname my-ns --external-name bar.com
```

选项

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

--external-name string

Service 对外的名称。

-h, --help

externalname 操作的帮助命令。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--save-config

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

--tcp strings

端口对可以指定为 "<端口>:<目标端口>"。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.3.8 ccictl create deployment

操作背景

创建指定名称的 Deployment。

```
ccictl create deployment NAME --image=image -- [COMMAND] [args...]
```

示例

```
# 创建一个名为 my-dep 的 Deployment，它将运行 busybox 镜像，Pod cpu/memory规格为1Core/2Gi
ccictl create deployment my-dep --image=busybox --pod-size-specs=1.0_2.00

# 创建一个带有命令的 Deployment，Pod cpu/memory规格为2Core/4Gi
ccictl create deployment my-dep --image=busybox --pod-size-specs=2.0_4.00 -- date

# 创建一个名为 my-dep 的 Deployment，它将运行 nginx 镜像并有 3 个副本，Pod cpu/memory规格为1Core/2Gi
ccictl create deployment my-dep --image=nginx --pod-size-specs=1.0_2.00 --replicas=3

# 创建一个名为 my-dep 的 Deployment，它将运行 busybox 镜像并公开端口 5701，Pod cpu/memory规格为
```

```
1Core/2Gi
ccictl create deployment my-dep --image=busybox --pod-size-specs=1.0_2.00 --port=5701

# 创建一个名为 my-dep 的 Deployment，Pod cpu/memory规格为1Core/2Gi，它将运行多个容器
ccictl create deployment my-dep --image=busybox:latest --pod-size-specs=1.0_2.00 --image=ubuntu:latest --image=nginx
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-h, --help
```

deployment 操作的帮助命令。

```
--image strings
```

要运行的镜像名称。Deployment 可以为多容器 Pod 设置多个镜像。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--port int32    默认值：-1
```

指定 Deployment 公开的 containerPort。

```
-r, --replicas int32    默认值：1
```

要创建的副本数。默认值为 1。

```
--save-config
```

如果为 true，当前对象的配置将被保存在其注解中。否则，注解将保持不变。 此标志在您希望后续对该对象执行 ccictl apply 时很有用。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

```
--pod-size-specs
```

指定pod实例规格。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.4 ccictl set

操作背景

配置应用程序资源。

这些命令可帮助您更改现有的应用程序资源。

```
ccictl set SUBCOMMAND
```

选项

```
-h, --help
```

关于 set 的帮助信息。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.4.1 ccictl set env

操作背景

更新 Pod 模板中的环境变量。

列举一个或多个 Pod 和 Pod 模板中的环境变量定义。 添加、更新或移除（在副本控制器或 Deployment 配置中的）一个或多个 Pod 模板中的容器环境变量定义。 查看或修改指定 Pod 或 Pod 模板中所有容器的环境变量定义，或者只查看与通配符匹配的那些环境变量定义。

如果在命令行上设置了 "--env -"，则可以使用标准的 env 语法从标准输入中读取环境变量。

可能的资源包括（不区分大小写）：

```
pod (po), deployment (deploy)
```

```
ccictl set env RESOURCE/NAME KEY_1=VAL_1 ... KEY_N=VAL_N
```

示例

```
# 使用新的环境变量更新 Deployment “registry”
ccictl set env deployment/registry STORAGE_DIR=/local

# 列举 Deployment “sample-build” 中定义的环境变量
ccictl set env deployment/sample-build --list

# 列举所有 Pod 中定义的环境变量
ccictl set env pods --all --list

# 以 YAML 格式输出修改后的 Deployment，但不更改服务器上的对象
ccictl set env deployment/sample-build STORAGE_DIR=/data -o yaml

# 更新项目中所有 Deployment 中的所有容器，为之添加 ENV=prod
ccictl set env deploy --all ENV=prod

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

# 从带前缀的 ConfigMap 中导入环境变量
ccictl set env --from=configmap/myconfigmap --prefix=MYSQL_ deployment/myapp

# 从 ConfigMap 中导入特定键
ccictl set env --keys=my-example-key --from=configmap/myconfigmap deployment/myapp

# 从所有 Deployment 配置中的容器 “c1” 中移除环境变量 ENV
ccictl set env deployments --all --containers="c1" ENV-

# 从磁盘上的 Deployment 定义中移除环境变量 ENV 并更新服务器上的 Deployment 配置
ccictl set env -f deploy.json ENV-

# 将某些本地 Shell 环境变量设置到服务器上的 Deployment 配置中
env | grep RAILS_ | ccictl set env -e - deployment/registry
```

选项

--all

如果为真，则选择指定资源类型的命名空间中的所有资源。

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

-c, --containers string　默认值："*"

所选 Pod 模板中要更改的容器名称 - 可以使用通配符。

-e, --env strings

为某个环境变量指定一个键值对列表，以设置到每个容器中。

-f, --filename strings

文件名、目录或文件 URL 组成的列表，用于标识要更新环境的资源。

--from string

要从中注入环境变量的资源的名称。

-h, --help

env 操作的帮助命令。

--keys strings

要从指定的资源中导入的、以英文逗号分隔的键的列表。

--list

如果为真，则以标准格式显示环境及所有变更。 当使用 `ccictl view env` 命令时，此标志将被移除。

--local

如果为真，`set env` 将不会与 API 服务器通信，而是在本地运行。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--overwrite　默认值：true

如果为真，允许环境被覆盖，否则拒绝要覆盖现有环境的更新。

--prefix string

要追加到变量名上的前缀。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

--resolve

如果为真，则在列出变量时显示 Secret 或 ConfigMap 引用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.4.2 ccictl set image

操作背景

更新资源的现有容器镜像。

可能的资源包括（不区分大小写）：

```
pod (po), deployment (deploy)
```

命令的格式如下：

```
ccictl set image (-f FILENAME | TYPE NAME) CONTAINER_NAME_1=CONTAINER_IMAGE_1 ... CONTAINER_NAME_N=CONTAINER_IMAGE_N
```

示例

```
# 将 Deployment 的 nginx 容器镜像设置为 “nginx:1.9.1”，并将其 busybox 容器镜像设置为 “busybox”
ccictl set image deployment/nginx busybox=busybox nginx=nginx:1.9.1

# 更新所有 Deployment 的 nginx 容器镜像为 “nginx:1.9.1”
ccictl set image deployments nginx=nginx:1.9.1 --all

# 更新 Deployment abc 的所有容器镜像为 "nginx:1.9.1"
ccictl set image deploy abc *=nginx:1.9.1

# 使用本地文件更新 nginx 容器镜像，并以 YAML 格式打印结果，但不向服务器发出请求
ccictl set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml
```

选项

```
--all
```

如果为真，则选择指定资源类型的命名空间中的所有资源。

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-f, --filename strings
```

文件名、目录或文件 URL 组成的列表，用于标识要更新环境的资源。

```
-h, --help
```

image 操作的帮助命令。

```
--local
```

如果为真，`set image` 将不会与 API 服务器通信，而是在本地运行。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.4.3 ccictl set resources

操作背景

为定义 Pod 模板的任一资源指定计算资源要求（CPU、内存）。 如果 Pod 被成功调度，将保证获得所请求的资源量，但可以在某一瞬间达到其指定的限制值。

对于每类计算资源，如果只指定限制值而省略请求值，则请求值将被默认设置为限制值。

可能的资源包括（不区分大小写）： 使用 "ccictl api-resources" 查看受支持资源的完整列表。

```
ccictl set resources (-f FILENAME | TYPE NAME) ([--limits=LIMITS & --requests=REQUESTS]
```

示例

```
# 将 Deployment nginx 中容器 nginx 的 CPU 限制设置为 "200m"， 将内存限制设置为 "512Mi"
ccictl set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# 为 nginx 中的所有容器设置资源请求和限制
ccictl set resources deployment nginx --limits=cpu=200m,memory=512Mi --
requests=cpu=100m,memory=256Mi

# 移除 nginx 中容器对资源的资源请求
ccictl set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=0,memory=0

# 打印基于本地清单更新 nginx 容器限制的结果（以 YAML 格式），不向服务器发送请求
ccictl set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --local -o yaml
```

选项

--all

如果为真，则选择指定资源类型的命名空间中的所有资源。

--allow-missing-template-keys　默认值：true

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-c, --containers string    默认值："*"
```

所选 Pod 模板中要更改的容器名称 - 可以使用通配符。

```
-f, --filename strings
```

文件名、目录或文件 URL 组成的列表，用于标识要更新环境的资源。

```
-h, --help
```

resources 操作的帮助命令。

```
--limits string
```

指定请求里容器的limit资源。例如，“cpu=100m,memory=256Mi”。 请注意，服务器端组件可能会根据服务器配置（例如 LimitRange）分配请求。

```
--local
```

如果为真，`set image` 将不会与 API 服务器通信，而是在本地运行。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--requests string
```

指定容器的资源请求。例如，“cpu=100m,memory=256Mi”。 请注意，服务器端组件可能会根据服务器配置（例如 LimitRange）分配请求。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.4.4 ccictl set selector

操作背景

支持某个资源设置选择算符。选择算符必须以字母或数字开头，可以包含字母、数字、连字符、点和下划线，最长为 63 个字符。 如果指定了 --resource-version，则更新将使用此资源版本，否则将使用现有的资源版本。

```
ccictl set selector (-f FILENAME | TYPE NAME) EXPRESSIONS [--resource-version=version]
```

约束与限制

- 目前仅支持在 Service 对象上设置选择算符。
- 如果资源在 set selector 调用之前已有选择算符，则新的选择算符将覆盖旧的选择算符。

示例

```
# 在创建 Service 之后设置选择算符
ccictl create service loadbalancer my-svc --tcp=8088:8088 --elb-id="xxx" -o yaml -nt1 | ccictl set selector --local -f - 'environ
ment=qa' -oyaml
```

选项

```
--all
```

如果为真，则选择指定资源类型的命名空间中的所有资源。

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-f, --filename strings
```

使用文件名来区分不同的资源。

```
-h, --help
```

selector 操作的帮助命令。

```
--local
```

如果为真，`set selector` 将不会与 API 服务器通信，而是在本地运行。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--resource-version string
```

如果非空，则只有在所给值是对象的当前资源版本时，选择算符更新才会成功。仅在指定单个资源时有效。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.5 ccictl explain

操作背景

描述各种资源的字段和结构。

此命令描述与每个被支持的 API 资源关联的字段。 这些字段通过一个简单的 JSONPath 标识符进行识别：

```
<类型>.<字段名>[.<字段名>]
```

有关每个字段的信息是以 OpenAPI 格式从服务器中检索而来的。

使用 "ccictl api-resources" 获取受支持的资源的完整列表。

```
ccictl explain TYPE [--recursive=FALSE|TRUE] [--api-version=api-version-group] [-o|--output=plaintext|plaintext-openapiv2]
```

示例

```
# 获取资源及其字段的文档
ccictl explain pods

# 获取资源中的所有字段
ccictl explain pods --recursive

# 获取被支持的 API 版本中 Deployment 的解释
ccictl explain deployments --api-version=apps/v1

# 获取资源中特定字段的文档
ccictl explain pods.spec.containers
```

选项

```
--api-version string
```

使用资源的给定的 API 版本（组/版本）。

```
-h, --help
```

explain 操作的帮助命令。

```
--recursive
```

如果为真，递归打印所有字段的名称。否则，打印可用字段及其描述。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.6 ccictl get

操作背景

显示一个或多个资源。

打印一张包含与指定资源相关的最重要信息的表格。 您可以使用标签选择算符（--selector 标志）来过滤列表。 如果所请求的资源类型是命名空间作用域的，您只会看到当前命名空间中的结果， 除非您传递 --all-namespaces 参数。

通过将输出指定为 “template” 并提供一个 Go 模板作为 --template 标志的值，您可以过滤所读取资源的属性。

使用 "ccictl api-resources" 获取受支持的资源的完整列表。

```
ccictl get [(-o|--output=)json|yaml|name|go-template|go-template-file|template|templatefile|jsonpath|
jsonpath-as-json|jsonpath-file|custom-columns|custom-columns-file|wide] (TYPE[.VERSION][.GROUP]
[NAME | -l label] | TYPE[.VERSION][.GROUP]/NAME ...) [flags]
```

示例

```
# 以 ps 输出格式列举所有 Pod
ccictl get pods

# 以 ps 输出格式列举所有 Pod，并提供更多信息（如PodIP）
ccictl get pods -o wide

# 以 ps 输出格式列举指定名称的单个 Pod
ccictl get po web

# 以 JSON 输出格式列举 "cci" API 组 "v2" 版本中的 Deployment
ccictl get deployments.v2.cci -o json

# 以 JSON 输出格式列举单个 Pod
ccictl get -o json pod web-pod-13je7

# 以 JSON 输出格式列举在 "pod.yaml" 中以 type 和 name 指定的 Pod
ccictl get -f pod.yaml -o json

# 仅返回指定 Pod 的 phase 值
ccictl get -o template pod/web-pod-13je7 --template={{.status.phase}}

# 在自定义列中列举资源信息
ccictl get pod test-pod -o custom-
columns=CONTAINER:.spec.containers[0].name,IMAGE:.spec.containers[0].image

# 以 ps 输出格式同时列举所有 Deployment 和服务
ccictl get deploy,services

# 按类型和名称列举一个或多个资源
ccictl get deploy/web service/frontend pods/web-pod-13je7

# 列出 “backend” 命名空间中的所有 Deployment
ccictl get deployments --namespace backend

# 列出所有命名空间中存在的所有 Pod
ccictl get pods --all-namespaces
```

选项

```
-A, --all-namespaces
```

如果存在此标志，则跨所有命名空间列举所请求的对象。 即使使用 --namespace 指定了命名空间，当前上下文中的命名空间也会被忽略。

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--chunk-size int    默认值：500
```

以块的形式返回大的列表，而不是一次性全部返回。设为 0 表示禁用。

```
--field-selector string
```

过滤所用的选择算符（字段查询），支持 '='、'==' 和 '!='。（例如 --field-selector key1=value1,key2=value2）。服务器针对每种类型仅支持有限数量的字段查询。

```
-f, --filename strings
```

文件名、目录或文件 URL 列表，用于标识要从服务器获取的资源。

-h, --help

get 操作的帮助命令。

--ignore-not-found

如果请求的对象不存在，此命令将返回退出码 0。

-L, --label-columns strings

接受一个用逗号分隔的标签列表，这些标签将被用作所打印表格中的不同列。 名称区分大小写。您也可以使用多个标志选项，例如 -L label1 -L label2...

--no-headers

当使用默认或自定义列输出格式时，不要打印标题（默认打印标题）。

-o, --output string

输出格式。可选值为：json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、 jsonpath-as-json、jsonpath-file、custom-columns、custom-columns-file、wide。参见自定义列 [**自定义列**]、golang 模板 [http://golang.org/pkg/text/template/#pkg-overview] 和 jsonpath 模板 [**JSONPath 支持**]。

--output-watch-events

使用 --watch 或 --watch-only 标志时输出监视事件对象。现有对象被输出为初始的 ADDED 事件。

--raw string

向服务器发送请求所用的原始 URI。使用 cliconfig 文件中指定的传输方式。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

--server-print　默认值：true

如果为 true，则令服务器返回适当的表格输出。支持扩展 API 和 CRD。

--show-kind

如果存在此标志，则列举所请求对象的资源类型。

--show-labels

打印时，将所有标签显示为最后一列（默认隐藏标签列）。

--sort-by string

如果非空，则使用此字段规约对列表类型进行排序。 字段规约表示为 JSONPath 表达式（例如 “{.metadata.name}”）。 由此 JSONPath 表达式指定的 API 资源中的字段必须是一个整数或字符串。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

```
-w, --watch
```

列举/获取请求的对象后，监视其变化。

```
--watch-only
```

监视所请求对象的变化，而不先列举/获取对象。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.7 ccictl edit

操作背景

用默认编辑器编辑资源。

- edit 命令允许您直接编辑可通过命令行工具检索的任何 API 资源。 它将打开由 EDITOR 环境变量定义的编辑器，或者回退到使用 Linux 的 "vi" 或 Windows 的 "notepad"。 当尝试打开编辑器时，它将首先尝试使用 "SHELL" 环境变量中定义的 shell。 如果未定义，将使用默认的 shell，在 Linux 中为 "/bin/bash"，在 Windows 中为 "cmd"。
- 您可以编辑多个对象，但一次只会应用一个更改。该命令接受文件名和命令行参数，尽管您指向的文件必须是以前保存的资源版本。
- edit 是通过用于获取资源的 API 版本完成的。要编辑特定 API 版本的资源，请完全限定资源、版本和组。
- 默认格式为 YAML。要以 JSON 格式进行编辑，请指定 "-o json"。
- --windows-line-endings 标志可用于强制 Windows 行结束，否则将使用操作系统的默认值。
- 如果更新时发生错误，将在磁盘上创建一个临时文件，其中包含未应用的更改。更新资源时最常见的错误是另一个编辑器更改了服务器上的资源。发生这种情况时， 您必须在应用到较新版本的资源进行更新，或更新临时保存的副本以包含最新的资源版本。

```
ccictl edit (RESOURCE/NAME | -f FILENAME)
```

示例

```
# 编辑名为 "registry" 的 Service
ccictl edit svc/registry

# 使用替代编辑器
EDITOR="nano" ccictl edit svc/registry

# 使用 cci/v2 API 格式编辑 JSON 中的 Deployment "mydeploy"
ccictl edit deployment.v2.cci/mydeploy -o json

# 在 YAML 中编辑 Deployment "mydeployment" 并将修改后的配置保存在其注解中
ccictl edit deployment/mydeployment -o yaml --save-config
```

选项

```
-f, --filename strings
```

用于编辑资源的文件名、目录或文件 URL 的列表。

-h, --help

关于 edit 的帮助信息。

-o, --output string

输出格式。可选值为： json、yaml。

--output-patch

如果资源被编辑，则输出补丁。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

--save-config

如果为 true，则当前对象的配置将被保存在其注解中。否则，注解将保持不变。 当您希望后续对此对象执行 `ccictl apply` 操作时，此标志很有用。

--validate string[="strict"] 默认值："strict"

必须是以下选项之一：strict（或 true）、warn、ignore（或 false）。 "true" 或 "strict" 将使用模式定义来验证输入，如果无效，则请求失败。 "false" 或 "ignore" 将不会执行任何模式定义检查，而是静默删除所有未知或重复的字段。

--windows-line-endings

默认为您所用平台原生的行结尾格式。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.8 ccictl delete

操作背景

基于文件名、标准输入、资源和名称，或基于资源和标签选择算符来删除资源。

接受 JSON 和 YAML 格式。只能指定一种类型的参数：文件名、资源加名称，或资源加标签选择算符。

某些资源（如 Pod）支持体面删除。这些资源定义了在强制终止之前的默认时长（宽限期）， 但您可以使用 --grace-period 标志覆盖该值，或传递 --now 将宽限期设置为 1。 由于这些资源通常代表集群中的实体，所以删除可能不会立即得到确认。 如果无法访问 API 服务器，则终止所用的时间可能比宽限期长得多。 要强制删除某资源，您必须指定 --force 标志。 注意：只有一部分资源支持体面删除。如果不支持体面删除，--grace-period 标志将被忽略。

重要提示：强制删除 Pod 不会等待确认 Pod 的进程已被终止， 这可能会导致直到节点检测到删除请求并完成体面删除之前，Pod 中的进程一直继续运行。 只有在您确定 Pod 已被终止或您的应用可以容忍同时运行相同 Pod 的多个副本时，才可以强制删除 Pod。

请注意，删除命令不会检查资源版本，因此如果有人在您提交删除指令时提交了资源更新指令， 他们的更新请求将与剩余的资源一起丢失。

```
ccictl delete ([-f FILENAME] | TYPE [(NAME | -l label | --all)])
```

示例

```
# 使用 pod.json 中指定的类型和名称删除一个 Pod
ccictl delete -f ./pod.json

# 基于目录中的内容删除资源
ccictl delete -f dir/

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

# 基于传递到标准输入的 JSON 中的类型和名称删除一个 Pod
cat pod.json | ccictl delete -f -

# 删除名称为 "baz" 和 "foo" 的 Pod 和 Service
ccictl delete pod,service baz foo

# 删除打了标签 name=myLabel 的 Pod 和 Service
ccictl delete pods,services -l name=myLabel

# 以最小延迟删除一个 Pod
ccictl delete pod foo --now

# 强制删除一个 Pod
ccictl delete pod foo --force

# 删除所有 Pod
ccictl delete pods --all
```

选项

```
--all
```

删除指定资源类型的命名空间中的所有资源。

```
-A, --all-namespaces
```

如果存在，则列举所有命名空间中请求的对象。 即使使用 --namespace 指定，当前上下文中的命名空间也会被忽略。

```
--cascade string[="background"]    默认值："background"
```

必须是 "background"、"orphan" 或 "foreground"。 选择依赖项（例如，由 Deployment 创建的 Pod）的删除级联策略， 默认为 background。

```
--field-selector string
```

过滤所用的选择算符（字段查询），支持 '='、'==' 和 '!='。（例如 --field-selector key1=value1,key2=value2）。服务器针对每种类型仅支持有限数量的字段查询。

```
-f, --filename strings
```

包含要删除的资源的文件名。

```
--force
```

如果为真，则立即从 API 中移除资源并略过体面删除处理。 请注意，立即删除某些资源可能会导致不一致或数据丢失，并且需要确认操作。

```
--grace-period int    默认值：-1
```

指定给资源的体面终止时间（以秒为单位）。 如果为负数则忽略，为 1 表示立即关闭。 仅当 --force 为真（强制删除）时才可以设置为 0。

```
-h, --help
```

delete 操作的帮助命令。

--ignore-not-found

将 “resource not found” 视为成功删除。当指定 --all 参数时，默认值为 “true” 。

--now

如果为 true，资源将被标记为立即关闭（等同于 --grace-period=1 ）。

-o, --output string

输出模式。使用 “-o name” 以获得更简短的输出（resource/name ）。

--raw string

向服务器发送 DELETE 请求所用的原始 URI。使用 cliconfig 文件中指定的传输方式。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2 ）。匹配的对象必须满足所有指定的标签约束。

--timeout duration

放弃删除之前等待的时间长度，为 0 表示根据对象的大小确定超时。

--wait 默认值：true

如果为 true，则等待资源消失后再返回。此参数会等待终结器被清空。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.9 ccictl rollout

操作背景

管理一个或多个资源的上线。

有效的资源类型如下：

deployments

```
ccictl rollout SUBCOMMAND
```

示例

```
# 回滚到先前的 Deployment 版本
ccictl rollout undo deployment/abc

# 检查 Deployment 的部署状态
ccictl rollout status deploy/foo

# 重启 Deployment
ccictl rollout restart deployment/abc
```

```
# 重启带有 'app=nginx' 标签的 Deployment
ccictl rollout restart deployment --selector=app=nginx
```

选项

```
-h, --help
```

rollout 命令的帮助信息。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.9.1 ccictl rollout history

操作背景

查看以前上线的修订版本和配置。

```
ccictl rollout history (TYPE NAME | TYPE/NAME) [flags]
```

示例

```
# 查看 Deployment 的上线历史记录
ccictl rollout history deployment/abc

# 查看 Deployment 修订版本 3 的详细信息
ccictl rollout history deployment/abc --revision=3
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-f, --filename strings
```

文件名、目录或文件 URL 列表，用于标识要从服务器获取的资源。

```
-h, --help
```

关于 history 的帮助信息。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--revision int
```

查看详细信息，包括指定修订版本的 Pod 模板。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.9.2 ccictl rollout restart

操作背景

重启资源。

资源将重新开始上线。

```
ccictl rollout restart RESOURCE
```

示例

```
# 重启 test-namespace 命名空间下的所有 Deployment
ccictl rollout restart deployment -n test-namespace

# 重启 Deployment
ccictl rollout restart deployment/nginx

# 重启带有标签 app=nginx 的 Deployment
ccictl rollout restart deployment --selector=app=nginx
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-f, --filename strings
```

文件名、目录或文件 URL 列表，用于标识要从服务器获取的资源。

```
-h, --help
```

关于 restart 的帮助信息。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.9.3 ccictl rollout status

操作背景

显示上线的状态。

默认情况下，"rollout status" 将监视最新的上线状态，直到它完成。 如果您不想等待 rollout 完成，则可以使用 --watch=false。请注意，如果中间开始了新的上线动作，则 "rollout status" 将继续监视最新修订版本。如果您想 watch 特定的修订版本并在它被另一个修订覆盖时中止， 请使用 --revision=N，其中 N 是您需要监视的修订版本。

```
ccictl rollout status (TYPE NAME | TYPE/NAME) [flags]
```

示例

```
# 监视部署的上线状态
ccictl rollout status deployment/nginx
```

选项

```
-f, --filename strings
```

文件名、目录或文件 URL 列表，用于标识要从服务器获取的资源。

```
-h, --help
```

关于 status 的帮助信息。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--revision int
```

固定到特定修订版本以显示其状态。默认为 0（最后修订版本）。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--timeout duration
```

结束监视之前等待的时间长度，零表示永不结束。任何其他值都应包含相应的时间单位（例如 1s、2m、3h）。

```
-w, --watch     Default: true
```

监视上线状态直至上线完成。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.9.4 ccictl rollout undo

操作背景

回滚到之前上线的版本。

```
ccictl rollout undo (TYPE NAME | TYPE/NAME) [flags]
```

示例

```
# 回滚到上一个 Deployment 的上一次部署状态
ccictl rollout undo deployment/abc

# 回滚到 Deployment 的修订版本 3
ccictl rollout undo deploy/abc --to-revision=3
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
-f, --filename strings
```

文件名、目录或文件 URL 列表，用于标识要从服务器获取的资源。

```
-h, --help
```

关于 undo 的帮助信息。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

```
--to-revision int
```

要回滚到的修订版本。默认为 0（最新修订版本）。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.10 ccictl describe

操作背景

显示某个特定资源或某组资源的细节。

打印所选资源的详细描述，包括事件或控制器等相关资源。 您可以通过名称选择单个对象，选择该类型的所有对象，提供名称前缀或标签选择算符。例如：

```
ccictl describe TYPE NAME_PREFIX
```

这条命令将首先检查与 TYPE 和 NAME_PREFIX 的精确匹配。 如果不存在这样的资源，它将输出名称以 NAME_PREFIX 为前缀的所有资源的细节。

使用 "ccictl api-resources" 获取受支持资源的完整列表。

```
ccictl describe (-f FILENAME | TYPE [NAME_PREFIX | -l label] | TYPE/NAME)
```

示例

```
# 描述一个节点
ccictl describe deploy example-deploy

# 描述一个 Pod
ccictl describe pods/nginx

# 描述在 "pod.json" 中通过类别和名称标识的 Pod
ccictl describe -f pod.json

# 描述所有 Pod
ccictl describe pods

# 描述带标签 name=myLabel 的 Pod
ccictl describe pods -l name=myLabel

# 描述由 “frontend” Deployment 管理的所有 Pod
ccictl describe pods frontend
```

选项

```
-A, --all-namespaces
```

如果存在此标志，则跨所有命名空间列举所请求的对象。 即使使用 --namespace 指定了命名空间，当前上下文中的命名空间也会被忽略。

```
--chunk-size int    默认值：500
```

以块的形式返回大的列表，而不是一次性全部返回。设为 0 表示禁用。

```
-f, --filename strings
```

文件名、目录或文件 URL 的列表，包含要描述的资源。

```
-h, --help
```

describe 操作的帮助命令。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
-l, --selector string
```

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.11 ccictl logs

操作背景

打印 Pod 或指定资源中某个容器的日志。如果 Pod 只有一个容器，则容器名称是可选的。

```
ccictl logs [-f] [-p] (POD | TYPE/NAME) [-c CONTAINER]
```

示例

```
# 返回只有一个容器的 nginx Pod 中的快照日志
ccictl logs nginx

# 从 nginx Pod 返回快照日志，每行前面加上来源的 Pod 和容器名称
ccictl logs nginx --prefix

# 从 nginx Pod 返回快照日志，限制输出为 500 字节
ccictl logs nginx --limit-bytes=500

# 从 nginx Pod 返回快照日志，等待其启动运行最多 20 秒
ccictl logs nginx --pod-running-timeout=20s

# 返回有多个容器的 nginx Pod 中的快照日志
ccictl logs nginx --all-containers=true

# 返回带 app=nginx 标签定义的 Pod 中所有容器的快照日志
ccictl logs -l app=nginx --all-containers=true

# 返回带 app=nginx 标签定义的 Pod 中容器的快照日志，限制并发日志请求为 10 个 Pod
ccictl logs -l app=nginx --max-log-requests=10

# 返回 web-1 Pod 中之前终止的 ruby 容器日志的日志
ccictl logs -p -c ruby web-1

# 开始从 nginx Pod 流式传输日志，即使发生错误也继续
ccictl logs nginx -f --ignore-errors=true

# 开始流式传输 web-1 Pod 中 ruby 容器的日志
ccictl logs -f -c ruby web-1

# 开始流式传输带 app=nginx 标签定义的 Pod 中所有容器的日志
ccictl logs -f -l app=nginx --all-containers=true

# 仅显示 nginx Pod 的最近 20 行输出
ccictl logs --tail=20 nginx

# 显示 nginx Pod 在过去一小时内写入的所有日志
ccictl logs --since=1h nginx

# 显示从 2024 年 8 月 30 日 06:00:00 UTC 开始 nginx Pod 中所有带时间戳的日志
ccictl logs nginx --since-time=2024-08-30T06:00:00Z --timestamps=true

# 显示 nginx pod 的日志，跳过kubelet证书验证
ccictl logs --insecure-skip-tls-verify-backend nginx

# 返回 nginx Deployment 的 nginx-1 容器的快照日志
ccictl logs deployment/nginx -c nginx-1
```

选项

--all-containers

获取 Pod 中所有容器的日志。

-c, --container string

打印指定容器的日志。

-f, --follow

指定日志是否应以流式传输。

-h, --help

logs 操作的帮助命令。

--ignore-errors

如果在监视/跟随 Pod 日志，则允许出现任何非致命的错误。

--insecure-skip-tls-verify-backend

跳过请求日志来源的 kubelet 的身份验证。从理论上讲，攻击者可能会提供无效的日志内容。 如果您的 kubelet 提供的证书已过期，您可能需要使用此参数。

--limit-bytes int

要返回的日志的最大字节数。默认为无限制。

--max-log-requests int 默认值：5

指定使用选择算符时要遵循的最大并发日志数。默认值为 5。

--pod-running-timeout duration 默认值：20s

等待至少一个 Pod 运行的时长（例如 5s、2m 或 3h，大于零）。

--prefix

在每行日志前添加日志来源（Pod 名称和容器名称）的前缀。

-p, --previous

如果为 true，则打印 Pod 中容器的前一个实例的日志（如果存在）。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

--since duration

仅返回比相对时长更新的日志，如 5s、2m 或 3h。 默认返回所有日志。只能使用 since-time 和 since 之一。

--since-time string

仅返回特定日期（RFC3339）之后的日志。默认返回所有日志。 只能使用 since-time 和 since 之一。

--tail int 默认值：-1

要显示的最近日志文件的行数。不带选择算符时默认为 -1 将显示所有日志行。 否则如果提供了选择算符，则为 10。

--timestamps

在日志输出的每一行中包含时间戳。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.12 ccictl exec

操作背景

在容器中执行命令。

```
ccictl exec (POD | TYPE/NAME) [-c CONTAINER] [flags] -- COMMAND [args...]
```

示例

```
# 在 Pod mypod 中执行 'date' 命令获取输出，默认在第一个容器中执行
ccictl exec mypod -- date

# 在 Pod mypod 的 ruby-container 容器中执行 'date' 命令并获取输出
ccictl exec mypod -c ruby-container -- date

# 切换到原始终端模式；从 Pod mypod 将 stdin 发送到 ruby-container 中的 'bash'，并将 stdout/stderr 从 'bash' 发送回客户端
ccictl exec mypod -c ruby-container -i -t -- bash -il

# 在 Pod mypod 的第一个容器中列出 /usr 的内容，并按修改时间排序
# 如果您要在 Pod 中执行的命令具有任何与 ccictl 本身重叠的标志（例如 -i），则必须使用两个破折号（--）来分隔命令的标志/参数
# 另请注意，不要用引号括住您的命令及其标志/参数，除非这是您正常执行它的方式（即执行 ls -t /usr，而不是 "ls -t /usr"）
ccictl exec mypod -i -t -- ls -t /usr
```

选项

```
-c, --container string
```

容器名称。 如果省略，则使用 kubectl.kubernetes.io/default-container 注解来选择要挂接的容器， 否则将选择 Pod 中的第一个容器。

```
-f, --filename strings
```

用于在资源中执行的文件。

```
-h, --help
```

关于 exec 的帮助信息。

```
--pod-running-timeout duration    默认：1m0s
```

等待至少一个 Pod 运行的时间长度（例如 5 秒、2 分钟或 3 小时，大于零）。

```
-q, --quiet
```

仅打印远程会话的输出。

```
-i, --stdin
```

将 stdin 传递给容器。

```
-t, --tty
```

Stdin 是一个 TTY。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.13 ccictl apply

操作背景

基于文件名或标准输入将配置应用于资源。必须指定资源名称。如果资源尚不存在，则资源会被创建。 若要使用 apply 命令，最初创建资源时应始终使用 apply 或 create --save-config。

支持 JSON 和 YAML 格式。

```
ccictl apply -f FILENAME
```

示例

```
# 将 pod.json 中的配置应用到 Pod
ccictl apply -f ./pod.json

# 应用来自目录中的资源
ccictl apply -f dir/

# 将传递到 stdin 的 JSON 应用到 Pod
cat pod.json | ccictl apply -f -

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

选项

```
--all
```

选择指定资源类型的命名空间中的所有资源。

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--cascade string[="background"]    默认值："background"
```

必须是 "background"、"orphan" 或 "foreground"。 选择依赖项（例如，由 Deployment 创建的 Pod）的删除级联策略， 默认为 background。

```
-f, --filename strings
```

包含了待应用的配置信息的文件。

```
--force
```

如果为真，则立即从 API 中移除资源并略过体面删除处理。 请注意，立即删除某些资源可能会导致不一致或数据丢失，并且需要确认操作。

```
--grace-period int    默认值：-1
```

指定给资源的体面终止时间（以秒为单位）。 如果为负数则忽略，为 1 表示立即关闭。 仅当 --force 为真（强制删除）时才可以设置为 0。

```
-h, --help
```

apply 操作的帮助命令。

```
--openapi-patch    默认值：true
```

如果为真，则当 openapi 存在且资源可在 openapi 规范中找到时，使用 openapi 计算 diff。 否则，回退到使用内置类型。

-o, --output string

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

--overwrite 默认值：true

使用修改后的配置中的值自动解决修改后的配置与实时配置之间的冲突。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！ ='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

--template string

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

--timeout duration

放弃删除之前等待的时间长度，为 0 表示根据对象的大小确定超时。

--validate string[="strict"] 默认值："strict"

必须是以下选项之一：strict（或 true）、warn、ignore（或 false）。 "true" 或 "strict" 将使用模式定义来验证输入，如果无效，则请求失败。"false" 或 "ignore" 将不会执行任何模式定义检查，而是静默删除所有未知或重复的字段。

--wait

如果为真，则等待资源消失后再返回。此参数会等待终结器被清空。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.13.1 ccictl apply edit-last-applied

操作背景

使用默认编辑器编辑资源的最新的 last-applied-configuration 注解。

- edit-last-applied 命令允许您直接编辑可以通过命令行工具检索的任何 API 资源。 它将打开由 EDITOR 环境变量定义的编辑器，或者在 Linux 上默认使用 "vi" 或在 Windows 上默认使用 "notepad"。 您可以编辑多个对象，不过所做的更改只能是逐个被应用的。此命令接受文件名以及命令行参数，但您指向的文件必须是资源的先前保存的版本。
- 默认格式为 YAML。若要以 JSON 格式编辑，请指定 -o json。
- 标志 --windows-line-endings 可用于强制使用 Windows 风格的行尾，否则将使用操作系统的默认设置。

- 如果在更新过程中发生错误，则会在磁盘上创建一个包含未被应用的变更的临时文件。 更新资源时最常见的错误是另一个编辑者更改了服务器上的资源，发生这种情况时，您必须将更改应用于资源的较新版本， 或更新临时保存的副本以包含最新的资源版本。

```
ccictl apply edit-last-applied (RESOURCE/NAME | -f FILENAME)
```

示例

```
# 在 YAML 中按类型/名称编辑 last-applied-configuration 注解
ccictl apply edit-last-applied deployment/nginx

# 通过 JSON 文件编辑 last-applied-configuration 注解
ccictl apply edit-last-applied -f deploy.yaml -o json
```

选项

```
-f, --filename strings
```

用于编辑资源的文件名、目录或文件 URL 的列表。

```
-h, --help
```

关于 edit-last-applied 的帮助信息。

```
-o, --output string
```

输出格式。可选值为： json、yaml。

```
-R, --recursive
```

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

```
--validate string[="strict"]    默认值："strict"
```

必须是以下选项之一：strict（或 true）、warn、ignore（或 false）。 "true" 或 "strict" 将使用模式定义来验证输入，如果无效，则请求失败。"false" 或 "ignore" 将不会执行任何模式定义检查，而是静默删除所有未知或重复的字段。

```
--windows-line-endings
```

默认为您所用平台本地的行结尾格式。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.13.2 ccictl apply set-last-applied

操作背景

设置 last-applied-configuration 注解使之与某文件内容相匹配。 这会导致 last-applied-configuration 被更新，就像运行了 ccictl apply -f <file> 一样， 但是不会更新对象的任何其他部分。

```
ccictl apply set-last-applied -f FILENAME
```

示例

```
# 设置资源的 last-applied-configuration，使之与某文件内容相同
ccictl apply set-last-applied -f deploy.yaml
```

```
# 针对目录中的每一个配置文件执行 set-last-applied 操作
ccictl apply set-last-applied -f path/

# 设置资源的 last-applied-configuration 注解，使之与某文件内容匹配；如果该注解尚不存在，则会被创建。
ccictl apply set-last-applied -f deploy.yaml --create-annotation=true
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--create-annotation
```

如果当前的对象没有 'last-applied-configuration' 注解，将该注解会被创建。

```
-f, --filename strings
```

包含 last-applied-configuration 注解的文件的文件名、目录或 URL 的列表。

```
-h, --help
```

关于 set-last-applied 的帮助信息。

```
-o, --output string
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--template string
```

当指定 `-o=go-template` 、`-o=go-template-file` 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.13.3 ccictl apply view-last-applied

操作背景

根据所给类别/名称或文件来查看最新的 last-applied-configuration 注解。

默认输出将以 YAML 格式打印到标准输出。您可以使用 -o 选项来更改输出格式。

```
ccictl apply view-last-applied (TYPE [NAME | -l label] | TYPE/NAME | -f FILENAME)
```

示例

```
# 根据所给类别/名称以 YAML 格式查看 last-applied-configuration 注解
ccictl apply view-last-applied deployment/nginx

# 根据所给文件以 JSON 格式查看 last-applied-configuration 注解
ccictl apply view-last-applied -f deploy.yaml -o json
```

选项

```
--all
```

选择指定资源类型的命名空间中的所有资源。

-f, --filename strings

包含 last-applied-configuration 注解的文件的文件名、目录或 URL 的列表。

-h, --help

view-last-applied 操作的帮助命令。

-o, --output string 默认值："yaml"

输出格式。必须是 yaml 或 json 之一。

-R, --recursive

递归处理在 -f、--filename 中给出的目录。当您想要管理位于同一目录中的相关清单时很有用。

-l, --selector string

过滤所用的选择算符（标签查询），支持 '='、'==' 和 '！='。（例如 -l key1=value1,key2=value2）。匹配的对象必须满足所有指定的标签约束。

ccictl选项亦可在子命令中生效，列表如下：

父命令ccictl选项列表

1.3.14 ccictl api-resources

操作背景

打印服务器支持的 API 资源。

```
ccictl api-resources [flags]
```

示例

```
# 打印服务器支持的 API 资源
ccictl api-resources

# 打印支持的 API 资源，但包含更多信息
ccictl api-resources -o wide

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

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

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

# 打印特定 APIGroup 支持的 API 资源
ccictl api-resources --api-group=cci
```

选项

-api-group string

限制为指定 API 组中的资源。

--cached

如果可用，将使用缓存的资源列表。

-h, --help

关于 api-resources 的帮助信息。

```
--namespaced    默认值：true
```

如果为false，则返回非命名空间作用域的资源，否则默认返回命名空间作用域的资源。

```
--no-headers
```

当使用默认或自定义列输出格式时，不要打印标题（默认打印标题）。

```
-o, --output string
```

输出格式，可选值为：wide、name。

```
--sort-by string
```

如果非空，则使用指定字段对资源列表进行排序，此字段可以是 "name" 或 "kind"。

```
--verbs strings
```

筛选支持指定动词的资源。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.15 ccictl api-versions

操作背景

以 “group/version” 的形式打印服务器支持的 API 版本。

```
ccictl api-versions
```

示例

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

选项

```
-h, --help
```

查看关于 api-versions 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16 ccictl config

操作背景

使用 "ccictl config set current-context my-context" 等子命令修改 cliconfig 文件。

加载顺序遵循以下规则：

- 如果设置了 --cliconfig 标志，则仅加载该文件。该标志只能设置一次，并且不会发生合并。

- 如果设置了 $CLICONFIG 环境变量，则将其用作路径列表（系统的正常路径分隔规则），这些路径会被合并。 当某个值被修改时，也会在定义这部分内容的文件中修改此值。当某个值被创建时，也会在存在的第一个文件中创建此值。 如果链中不存在文件，则它会创建列表中的最后一个文件。

否则，将使用 ${HOME}/.kubev2/config，并且不会发生合并。

```
ccictl config SUBCOMMAND
```

选项

```
-h, --help
```

关于 config 的帮助信息。

```
--cliconfig string
```

使用特定的 cliconfig 文件

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.1 ccictl config current-context

操作背景

显示当前上下文。

```
ccictl config current-context [flags]
```

示例

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

选项

```
-h, --help
```

关于 current-context 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.2 ccictl config get-contexts

操作背景

显示 cliconfig 文件中的一个或多个上下文。

```
ccictl config get-contexts [(-o|--output=)name)]
```

示例

```
# 列出 cliconfig 文件中的所有上下文
ccictl config get-contexts
```

```
# 描述 cliconfig 文件中指定上下文的详细信息
ccictl config get-contexts my-context
```

选项

-h, --help

关于 get-contexts 的帮助信息。

--no-headers

当使用默认或自定义列输出格式时，不要打印标题（默认打印标题）。

-o, --output string

输出格式。可选值为：name。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.3 ccictl config set-context

操作背景

在 cliconfig 中设置上下文条目。

指定已存在的属性名称将把新字段值与现有值合并。

```
ccictl config set-context [NAME | --current] [--cluster=cluster_nickname] [--user=user_nickname] [--
namespace=namespace]
```

示例

```
# 在 gce 上下文条目上设置用户字段，而不影响其他值
ccictl config set-context gce --user=cluster-admin
```

选项

--cluster string

cliconfig 中上下文条目的集群。

--current

修改当前上下文。

-h, --help

关于 set-context 的帮助信息。

--namespace string

cliconfig 中上下文条目的命名空间。

--user string

cliconfig 中上下文条目的用户。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.4 ccictl config delete-context

操作背景

从 cliconfig 中删除指定的上下文。

```
ccictl config delete-context NAME
```

示例

```
# 删除 example-context 集群的上下文
ccictl config delete-context example-context
```

选项

```
-h, --help
```

关于 delete-context 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.5 ccictl config rename-context

操作背景

重命名 cliconfig 文件中的上下文。

- CONTEXT_NAME 需要更改的上下文名称。
- NEW_NAME 是要设置的新名称。
- 注意：如果重命名的上下文是“当前上下文”，则该字段也将被更新。

```
ccictl config rename-context CONTEXT_NAME NEW_NAME
```

示例

```
# 将 cliconfig 文件中上下文 "old-name" 重命名为 "new-name"
ccictl config rename-context old-name new-name
```

选项

```
-h, --help
```

关于 rename-context 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.6 ccictl config use-context

操作背景

在 cliconfig 文件中设置当前上下文。

```
ccictl config use-context CONTEXT_NAME
```

示例

```
# 使用名为 exampleCtx1 上下文
kubectl config use-context exampleCtx1
```

选项

```
-h, --help
```

关于 use-context 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.7 ccictl config get-clusters

操作背景

显示 cliconfig 中定义的集群。

```
ccictl config get-clusters [flags]
```

示例

```
# 列出 ccictl 所知悉的集群
ccictl config get-clusters
```

选项

```
-h, --help
```

关于 get-clusters 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.8 ccictl config set-cluster

操作背景

设置 cliconfig 中的集群条目。

指定已存在的属性名称将把新字段值与现有值合并。

```
ccictl config set-cluster NAME [--server=server] [--certificate-authority=path/to/certificate/authority] [--insecure-skip-tls-verify=true] [--tls-server-name=example.com]
```

示例

```
# 仅设置 e2e 集群条目上的 server 字段，不涉及其他值
ccictl config set-cluster e2e --server=https://1.2.3.4

# 在 e2e 集群条目中嵌入证书数据
ccictl config set-cluster e2e --embed-certs --certificate-authority=~/.kubev2/e2e/ca.crt

# 禁用 e2e 集群条目中的证书检查
ccictl config set-cluster e2e --insecure-skip-tls-verify=true
```

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

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

选项

```
--certificate-authority string
```

cliconfig 中集群条目的证书颁发机构文件的路径。

```
--embed-certs tristate[=true]
```

在 cliconfig 中嵌入集群条目的证书。

```
-h, --help
```

关于 set-cluster 的帮助信息。

```
--insecure-skip-tls-verify tristate[=true]
```

设置 cliconfig 中的集群条目的 insecure-skip-tls-verify 字段。

```
--proxy-url string
```

cliconfig 中集群条目的代理地址。

```
--server string
```

cliconfig 中集群条目的 server 字段。

```
--tls-server-name string
```

cliconfig 中的集群条目的 tls-server-name 字段。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.9 ccictl config delete-cluster

操作背景

从 cliconfig 中删除指定的集群。

```
ccictl config delete-cluster NAME
```

示例

```
# 删除 example-cluster 集群
ccictl config delete-cluster example-cluster
```

选项

```
-h, --help
```

关于 delete-cluster 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.10 ccictl config get-users

操作背景

显示 cliconfig 中定义的用户。

```
ccictl config get-users [flags]
```

示例

```
# 列出 ccictl 知悉的用户
ccictl config get-users
```

选项

```
-h, --help
```

关于 get-users 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.11 ccictl config delete-user

操作背景

从 cliconfig 中删除指定用户。

```
ccictl config delete-user NAME
```

示例

```
# 删除 user1 用户
ccictl config delete-user user1
```

选项

```
-h, --help
```

关于 delete-user 的帮助信息。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.12 ccictl config set-credentials

操作背景

在 cliconfig 中设置用户条目。

- 指定已存在的属性名称将把新字段值与现有值合并。
  客户端证书标志：--client-certificate=certfile --client-key=keyfile
  持有者令牌标志：--token=bearer_token
  基本身份验证标志：--username=basic_user --password=basic_password

- 持有者令牌和基本身份验证是互斥的（不可同时使用）。

```
ccictl config set-credentials NAME [--client-certificate=path/to/certfile] [--client-key=path/to/keyfile] [--token=bearer_token] [--username=basic_user] [--password=basic_password] [--auth-provider=provider_name] [--auth-provider-arg=key=value] [--exec-command=exec_command] [--exec-api-version=exec_api_version] [--exec-arg=arg] [--exec-env=key=value]
```

示例

```
# 仅设置 "cluster-admin" 条目上的 "client-key" 字段，不涉及其他值
ccictl config set-credentials cluster-admin --client-key=~/.kubev2/admin.key

# 为 "cluster-admin" 条目设置基本身份验证
ccictl config set-credentials cluster-admin --username=admin --password=uXFGweU9l35qcif

# 在 "cluster-admin" 条目中嵌入客户端证书数据
ccictl config set-credentials cluster-admin --client-certificate=~/.kubev2/admin.crt --embed-certs=true

# 为 "cluster-admin" 条目启用 IAM 身份认证提供程序
ccictl config set-credentials cluster-admin --auth-provider=iam --auth-provider-arg=iam-endpoint=example.com

# 删除 "cluster-admin" 条目的 IAM 身份验证提供程序的 "iam-endpoint" 配置值
ccictl config set-credentials cluster-admin --auth-provider=iam --auth-provider-arg=iam-endpoint-

# 为 "cluster-admin" 条目启用新的 exec 认证插件
ccictl config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-api-version=client.authentication.k8s.io/v1beta1

# 为 "cluster-admin" 条目定义新的 exec 认证插件参数
ccictl 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
ccictl config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=key2=val2

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

选项

```
--auth-provider string
```

cliconfig 中用户条目的身份验证提供程序。

```
--auth-provider-arg strings
```

身份验证提供程序参数，'key=value' 格式。

```
--client-certificate string
```

cliconfig 中用户条目的客户端证书文件路径。

```
--client-key string
```

cliconfig 中用户条目的客户端密钥文件路径。

```
--embed-certs tristate[=true]
```

在 cliconfig 中嵌入用户条目的客户端证书/密钥。

```
--exec-api-version string
```

cliconfig 中用户条目的 exec 凭据插件的 API 版本。

```
--exec-arg strings
```

cliconfig 中用户条目的 exec 凭据插件命令的新参数。

```
--exec-command string
```


cliconfig 中用户条目的 exec 凭据插件命令。

```
--exec-env strings
```

exec 凭证插件的环境变量，'key=value' 格式。

```
-h, --help
```

关于 set-credentials 的帮助信息。

```
--password string
```

cliconfig 中用户条目的密码。

```
--token string
```

cliconfig 中用户条目的 token。

```
--username string
```

cliconfig 中用户条目的用户名。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.16.13 ccictl config view

操作背景

显示合并的 cliconfig 配置或指定的 cliconfig 文件。

可以使用 --output jsonpath={...} 通过 jsonpath 表达式提取特定值。

```
ccictl config view [flags]
```

示例

```
# 显示合并的 cliconfig 设置
ccictl config view

# 显示合并的 cliconfig 设置、原始证书数据和公开的密钥
ccictl config view --raw

# 获取 e2e 用户的密码
ccictl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

选项

```
--allow-missing-template-keys    默认值：true
```

如果为 true，在模板中字段或映射键缺失时忽略模板中的错误。 仅适用于 golang 和 jsonpath 输出格式。

```
--flatten
```

将生成的 cliconfig 文件扁平化为自包含的输出（对于创建可移植的 cliconfig 文件很有用）。

```
-h, --help
```

关于 view 的帮助信息。

```
--merge tristate[=true]    默认值：true
```

合并 cliconfig 文件的完整层次结构数据。

```
--minify
```

从输出中删除当前上下文未使用的所有信息。

```
-o, --output string    默认值："yaml"
```

输出格式。可选值为： json、yaml、name、go-template、go-template-file、template、templatefile、jsonpath、jsonpath-as-json、jsonpath-file。

```
--raw string
```

显示原始字节数据和敏感数据。

```
--template string
```

当 -o=go-template、-o=go-template-file 时使用的模板字符串或模板文件路径。 模板格式为 golang 模板 [http://golang.org/pkg/text/template/#pkg-overview]。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

1.3.17 ccictl version

操作背景

打印当前上下文的客户端和服务器版本信息。

```
ccictl version [flags]
```

示例

```
# 打印当前上下文的客户端和服务器版本
ccictl version
```

选项

```
-h, --help
```

关于版本的帮助信息

```
-o, --output string
```

`yaml` 或 `json` 之一。

ccictl选项亦可作为子命令的选项，列表如下：

父命令ccictl选项列表

2 使用 Terraform 管理 CCI 资源

2.1 Terraform 配置指南

什么是 Terraform

Terraform是一个开源的IT基础设施编排管理工具，通过Terraform您可以轻松的创建、管理、删除云容器实例资源，并对其进行版本控制。更多关于Terraform的介绍请参考Terraform产品介绍。

安装 Terraform

Terraform是以二进制可执行文件发布，您只需下载Terraform，然后将Terraform可执行文件所在目录添加到系统环境变量PATH中即可。

⚠ 注意

Terraform支持多个操作系统，如Linux、macOS、Windows等，您可以登录Terraform官网，下载对应操作系统的安装包，并将可执行Terraform文件添加至对应系统的环境变量中。

下文示例以Linux环境为例，进行Terraform安装说明。

1. 登录Terraform官网，下载对应的Linux操作系统安装包。
2. 在命令行中执行如下命令，解压安装包，赋予Terraform可执行权限，并放到PATH目录下。其中，*$PATH*为PATH路径（如/usr/local/bin），请替换为实际的值。
   ```
   unzip terraform_1.xx.x_
   chmod +x ./terraform
   mv ./terraform $PATH
   ```
3. 在命令行中执行如下命令验证配置路径是否正确。
   ```
   terraform -version
   ```
   如果回显如下则说明配置正确，Terraform可以运行。
   

认证与鉴权

使用Terraform管理云容器实例资源前，您需要获取AK（Access Key）/SK（Secret Key），并在Terraform上进行配置，从而认证鉴权。

您可以使用静态凭据或环境变量两种方式配置Terraform。

使用静态凭据的认证方式比较简单，但需要将AK/SK以明文的形式存储在配置文件中，存在密钥泄露的安全隐患。推荐您使用环境变量的方式进行认证。

- 静态凭据（Static credentials）

```
provider "huaweicloud" {
  region      = "cn-north-4"
  access_key  = "my-access-key"
  seccret_key = "my-secret-key"
}
```

region：区域，急需要创建管理那个区域的资源。您可以**单击此处**查询华为云支持的区域。

access_key：密钥ID，即AK。查询方法请参见**访问密钥**。

secret_key：访问密钥，即SK。查询方法请参见**访问密钥**。

- 环境变量（Environment variables）

将region、AK/SK等参数设置为环境变量的方式进行认证。

```
export HW_REGION_NAME="cn-north-4"
export HW_ACCESS_KEY="my-access-key"
export HW_SECRET_KEY="my-secret-key"
```

HW_REGION_NAME：区域，即需要创建管理那个区域的资源。您可以**单击此处**查询华为云支持的区域。

HW_ACCESS_KEY：密钥ID，即AK。查询方法请参见**访问密钥**。

HW_SECRET_KEY：访问密钥，即SK。查询方法请参见**访问密钥**。

初始化工作目录

1. 创建一个工作目录。

```
mkdir test
```

2. 在工作目录下创建“versions.tf”文件，指定华为云Provider的registry源和版本，文件内容如下：

```
terraform {
  required_providers {
    huaweicloud = {
      source = "huaweicloud/huaweicloud"
      version = ">=1.xx.xx" # 待加载provider的版本，可通过`=`指定需要加载的版本，通过`>=`指定需要加载的最小provider版本，优先加载最新版本。
    }
  }
}
```

说明

华为云Provider版本信息查询请参考【**华为云 provider**】。

3. 创建“main.tf”文件，配置华为云Provider，文件内容如下：

```
# Configure the HuaweiCloud Provider
provider "huaweicloud" {
  region      = "cn-north-4"
  access_key  = "my-access-key"
  secret_key  = "my-secret-key"
}
```

示例内容为HuaweiCloud Provider的配置，包含认证鉴权的内容，请根据**认证与鉴权**配置相关参数；如果使用环境变量方式认证鉴权，可以省略该部分内容。

4. 执行如下命令初始化。

```
terraform init
```

回显如下，首次执行时会下载HuaweiCloud Provider并安装。

```
[root@ecs-33d5-4f10 test]# terraform init
Initializing the backend...
Initializing provider plugins...
- Finding huaweicloud/huaweicloud versions matching ">= 1.20.0"...
- Installing huaweicloud/huaweicloud v1.76.5...
- Installed huaweicloud/huaweicloud v1.76.5 (self-signed, key ID 4FFE1736199213B8)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://developer.hashicorp.com/terraform/cli/plugins/signing
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

后续若需修改文件“versions.tf”或者“main.tf”文件配置信息，可通过如下命令进行升级。

```
terraform init -upgrade
```

2.2 Terraform 支持管理的 CCI 资源

表 2-1 Terraform 支持管理的华为云云容器实例资源范围

	create	update	get
namespace	√	×	√
network	√	√	√
deployment	√	√	√
pod	√	√	√
service	√	√	√
configmap	√	√	√
secret	√	√	√
hpa	√	√	√
pvc	√	√	√
pv	√	√	√
poolbinding	√	×	×
storageclasses	×	×	√

说明

- 更多关于云容器实例资源Terraform的详细使用介绍请参考【**terraform-provider-huaweicloud**】。
- 更多关于Terraform的语法介绍，请参考【**Huawei Cloud Terraform Docs**】。