ACP CLI (ac)

Getting Started with ACP CLI

Configuring ACP CLI

Usage of ac

Extending ACP CLI with Plugins

AC CLI Dev

AC CLI Administrator Command Reference

Upgrading Clusters

Getting Started with ACP CLI

TOC

- 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

With ACP CLI (ac), you can manage ACP platforms and clusters from a terminal. ACP CLI provides a kubectl-like experience optimized for ACP's centralized, proxy-based multi-cluster architecture.

ACP CLI is ideal in the following situations:

- Working with ACP platforms and multiple clusters from a unified interface
- Working directly with project source code Scripting ACP platform operations and automating workflows
- Managing projects while restricted by bandwidth resources and the web console is unavailable
- Managing applications across different ACP environments (production, staging, development)

Installation

Installing from Binary

You can install ACP CLI (ac) by downloading the binary for your operating system.

Follow these steps to download the correct package:

1. Open the Alauda Cloud download page ↗ in your browser.
2. Choose **CLI Tools** to enter the CLI download page.
3. Locate the **ACP CLI (ac)** section.
4. Download the binary that matches your operating system and CPU architecture (for example, `ac-linux-amd64`).

Installing ACP CLI on Linux

1. Complete the download steps above to obtain the Linux binary (for example, `ac-linux-amd64` or `ac-linux-arm64`).
2. Make the binary executable:

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

   Replace `ac-linux-amd64` with the filename you downloaded.
3. Move the binary into your PATH and rename it to `ac`:

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

   Adjust the filename if you downloaded a different variant.
4. Verify the installation:

   ```
   ac version
   ```

Installing ACP CLI on macOS

1. Complete the download steps above to obtain the macOS binary (for example, `ac-darwin-amd64` or `ac-darwin-arm64`).
2. Make the binary executable:

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

   Replace `ac-darwin-amd64` with the filename you downloaded.
3. Move the binary into your PATH and rename it to `ac`:

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

Adjust the filename if you downloaded a different variant.

4. Verify the installation:

```
ac version
```

Installing ACP CLI on Windows

1. Complete the download steps above to obtain the Windows binary (for example, `ac-windows-amd64.exe`).
2. Move the `ac-windows-amd64.exe` binary to a directory in your PATH and rename it to `ac.exe` if desired. Keep the original name if you prefer; just ensure the file's directory is in your PATH.
3. Verify the installation:

```
ac version
```

First Steps

Logging into ACP Platform

The `ac login` command is your entry point for connecting to ACP platforms. It handles authentication and automatically configures access to all available clusters.

Interactive Login

For the simplest experience, run `ac login` without parameters and follow the interactive prompts:

```
$ 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

You can also provide parameters directly:

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

Login Using Environment Variables

For automation and scripting, use environment variables:

```
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

Once logged in, ACP CLI provides convenient commands for daily operations:

View Current Status

Use `ac namespace` to see your current operational context:

```
$ 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

Switch between clusters within your current session:

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

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

Switch Namespaces

Change your active namespace:

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

Basic Resource Operations

Use standard kubectl commands to manage resources:

```
# List pods in current namespace
$ ac get pods

# Describe a specific pod
$ ac describe pod my-pod

# Get services across all namespaces
$ ac get services --all-namespaces

# Apply a configuration file
$ ac apply -f deployment.yaml
```

Managing Multiple Environments

For users working with multiple ACP platforms:

List all configured sessions:

```
$ ac config get-sessions
CURRENT   SESSION    PLATFORM                              USER
CLUSTERS
*         prod       https://acp.prod.example.com          user@example.c
om      3
          staging    https://staging.acp.example.com       user@example.c
om      2
```

Switch between platforms:

```
$ ac config use-session staging
Switched to session "staging".
Context "staging/global" activated.
```

Your First Application

Let's create and view a simple application to verify everything is working:

Create a Simple Pod

1. Create a basic pod configuration:

```
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. Apply the configuration:

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

View Application Status

1. List pods to see your application:

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

2. Get detailed information about the pod:

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

3. View pod logs:

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

Clean Up

Remove the test pod when finished:

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

Getting Help

Built-in Help System

ACP CLI provides comprehensive help at multiple levels:

General Help

Get an overview of all available commands:

```
ac help
```

Command-Specific Help

Get detailed help for any specific command:

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

Resource Documentation

Get information about Kubernetes resources:

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

Logging Out

When you're finished working or need to switch to different credentials, use the logout command:

```
$ 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
```

The logout command:

- Removes authentication tokens from your local configuration
- Cleans up all cluster and context entries for the session
- Revoke currently used tokens in the ACP
- Ensures no orphaned configuration remains

Configuring ACP CLI

TOC

Shell Completion
Enabling Tab Completion for Bash
Prerequisites
Procedure
Enabling Tab Completion for Zsh
Prerequisites
Procedure
Accessing kubeconfig by using ACP CLI
Prerequisites
Procedure
Multi-Cluster Configuration Handling
Security Considerations

Shell Completion

You can enable tab completion for the Bash or Zsh shells.

Enabling Tab Completion for Bash

After you install ACP CLI (ac), you can enable tab completion to automatically complete ac commands or suggest options when you press Tab. The following procedure enables tab completion for the Bash shell.

Prerequisites

- You must have ACP CLI (ac) installed.
- You must have the package bash-completion installed.

Procedure

1. Save the Bash completion code to a file:

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

2. Copy the file to `/etc/bash_completion.d/`:

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

   You can also save the file to a local directory and source it from your .bashrc file instead.

Tab completion is enabled when you open a new terminal.

Enabling Tab Completion for Zsh

After you install ACP CLI (ac), you can enable tab completion to automatically complete ac commands or suggest options when you press Tab. The following procedure enables tab completion for the Zsh shell.

Prerequisites

You must have ACP CLI (ac) installed.

Procedure

To add tab completion for ac to your .zshrc file, run the following command:

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

Tab completion is enabled when you open a new terminal.

Accessing kubeconfig by using ACP CLI

You can use ACP CLI (ac) to log in to your ACP platform and retrieve a kubeconfig file for accessing clusters from the command line. Unlike traditional single-cluster kubeconfig exports, ac login creates a comprehensive multi-cluster configuration through platform discovery.

Prerequisites

You have access to an ACP platform endpoint and valid authentication credentials.

Procedure

1. Log in to your ACP platform by running the following command:

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

   - `<platform-url>`: The base URL of the ACP platform (e.g., https://acp.prod.example.com↗)
   - `<session-name>`: A user-defined friendly name for this platform connection (e.g., "prod", "staging")

2. The login process automatically:

   - Authenticates with the ACP platform

- Discovers all accessible clusters in the platform
- Creates kubeconfig entries for all clusters with ACP-specific metadata
- Sets up a default context pointing to the global cluster

3. To export the configuration to a separate file, run:

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

4. Set the KUBECONFIG environment variable to point to the exported file:

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

5. Use ac to interact with your ACP clusters:

```
$ ac get nodes
```

Multi-Cluster Configuration Handling

ACP CLI login process creates a comprehensive kubeconfig structure that includes:

- **Multiple cluster entries**: One for each accessible cluster in the platform
- **Session metadata**: Platform URL, session name, and cluster descriptions stored in extension fields
- **Unified authentication**: Single user credential entry that works across all clusters in the platform
- **Intelligent naming**: Conflict-free naming conventions using `acp:<session>:<cluster>` format

Security Considerations

Important: The exported kubeconfig file contains authentication tokens that provide access to your ACP platform clusters.

- Store the file securely with appropriate file permissions

- Never commit kubeconfig files to version control systems
- Consider the token expiration and refresh requirements
- Use different session names for different environments (prod, staging, dev) to maintain clear separation

If you plan to reuse the exported kubeconfig file across sessions or machines, ensure it is stored securely and regularly synchronized with `ac config sync` to maintain current cluster lists.

Usage of ac and kubectl Commands

The Kubernetes command-line interface (CLI), kubectl, can be used to run commands against a Kubernetes cluster. Because ACP is a Kubernetes-compatible platform, you can use the supported kubectl binaries that ship with ACP CLI, or you can gain extended functionality by using the ac binary.

TOC

- The ac Binary
  - ACP Platform Integration
  - Intelligent Resource Routing
    - Resource Routing Example
  - Additional Commands
- The kubectl Binary

The ac Binary

The ac binary offers the same capabilities as the kubectl binary, but extends to natively support additional ACP platform features, including:

ACP Platform Integration

ACP CLI provides built-in support for ACP's centralized, proxy-based multi-cluster architecture:

- **Platform Authentication** - Built-in login command for secure authentication with ACP platforms
- **Session Management** - Multi-platform session management with commands like `ac login`, `ac config use-session`, and `ac logout`
- **Enhanced Configuration** - Additional commands like `ac config use-cluster` that make it easier to work with ACP multi-cluster environments

Intelligent Resource Routing

ACP CLI automatically routes platform-level resource types like `User` and `Project` to the global cluster, since these resources only exist at the platform level. This allows you to access them from any cluster context without manual switching. All other resources work normally with your current cluster context.

Resource Routing Example

```
# Current context points to workload cluster
$ ac config current-context
prod/workload-a

# User requests global resource - ACP CLI automatically routes to global cluster
$ 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

# User requests workload resource - operates on current cluster
$ ac get pods
NAME                     READY   STATUS    RESTARTS   AGE
my-app-7d4f8c9b6-xyz123  1/1     Running   0          2h
```

Additional Commands

ACP CLI includes additional commands that simplify ACP platform workflows:

- `ac login` - Authenticate to ACP platforms and configure multi-cluster access
- `ac logout` - End platform sessions and clean up configuration
- `ac config get-sessions` - List all configured ACP platform sessions
- `ac config use-session <session_name>` - Switch between ACP platforms
- `ac config use-cluster <cluster_name>` - Switch clusters within current session
- `ac namespace` - Enhanced namespace management with platform context display
- `ac config sync` - Synchronize configuration with platform state

The kubectl Binary

The kubectl binary is provided as a means to support existing workflows and scripts for new ACP CLI users coming from a standard Kubernetes environment, or for those who prefer to use the kubectl CLI. Existing users of kubectl can continue to use the binary to interact with Kubernetes primitives, with no changes required to the ACP platform.

For more information about kubectl, see the kubectl documentation↗.

Managing CLI Profiles

A CLI configuration file allows you to configure different profiles, or contexts, for use with ACP CLI (ac). A context consists of user authentication and ACP platform server information associated with a nickname.

TOC

- Convenient Configuration Management
  - Platform and Session Management
    - `ac login` - Authenticate and configure access to ACP platforms
    - `ac logout` - End platform sessions and clean up configuration
    - `ac config get-sessions` - List all configured ACP platform sessions
    - `ac config use-session <session_name>` - Switch between ACP platforms
  - Daily Operations
    - `ac config use-cluster <cluster_name>` - Switch clusters within current session
    - `ac namespace` - View current status and switch namespaces
    - `ac config sync` - Synchronize platform configuration
- Understanding ACP CLI Configuration Structure
  - ACP CLI-Enhanced kubeconfig Structure
  - Metadata Structure and Organization
    - Metadata-Based Identification
    - Naming Conventions
- Manual Configuration of CLI Profiles
  - Standard Configuration Commands

Example Manual Operations

Load and Merge Rules

Convenient Configuration Management

ACP CLI provides enhanced commands that make configuration management much easier than traditional kubeconfig manipulation. These commands are designed to work seamlessly with ACP's multi-cluster environment.

Platform and Session Management

`ac login` - Authenticate and configure access to ACP platforms

The `ac login` command serves as the primary entry point for establishing connections to ACP platforms. It authenticates users and automatically configures all necessary kubeconfig entries.

```
# Interactive login to an ACP platform
ac login https://prod.acp.com --name prod

# Login with specific cluster and namespace
ac login https://prod.acp.com --name prod --cluster workload-a --namespac
e my-app

# Login with environment variables (for automation)
AC_LOGIN_PLATFORM_URL=https://prod.acp.com AC_LOGIN_SESSION=prod AC_LOGIN
_USERNAME=user AC_LOGIN_PASSWORD=secret ac login
```

The login process:

1. Authenticates against the ACP platform
2. Discovers all accessible clusters within the platform
3. Creates cluster and user entries in your kubeconfig
4. Creates and activates a context:

- If `--cluster` is specified: creates context for that specific cluster
- If `--namespace` is specified: sets the namespace in the context
- If no cluster is specified: defaults to the global cluster
- Context name follows pattern: `<session_name>/<cluster_name>`

`ac logout` - End platform sessions and clean up configuration

```
# Log out from current platform session
ac logout

# Log out from a specific session
ac logout --session prod
```

The logout command removes all session-related configuration entries including clusters, users, and contexts.

`ac config get-sessions` - List all configured ACP platform sessions

```
ac config get-sessions
```

Example output:

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

This command displays:

- **CURRENT**: Indicates if the current context belongs to this session (marked with `*`)
- **SESSION**: Session name (user-defined during login)
- **PLATFORM**: Base platform URL

- **USER**: Authenticated username for the session
- **CLUSTERS**: Number of clusters available in this session

`ac config use-session <session_name>` - Switch between ACP platforms

```
# Switch to staging platform (defaults to global cluster)
ac config use-session staging

# Switch to specific cluster within a session
ac config use-session prod --cluster workload-a

# Switch with namespace specification
ac config use-session staging --cluster workload-b --namespace my-app
```

This command intelligently selects or creates appropriate contexts based on your session and cluster requirements.

Daily Operations

`ac config use-cluster <cluster_name>` - Switch clusters within current session

```
# Switch to workload cluster within current session
ac config use-cluster workload-a

# Create new context with specific namespace
ac config use-cluster workload-b --namespace my-app
```

This command finds or creates contexts for the specified cluster within your current platform session.

`ac namespace` - View current status and switch namespaces

Display current status:

```
ac namespace
```

Example output:

```
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
```

Switch namespace:

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

`ac config sync` - Synchronize platform configuration

```
# Sync current platform session
ac config sync

# Sync specific session
ac config sync --session prod

# Sync all sessions
ac config sync --all
```

The sync command refreshes your configuration with the latest information from ACP platforms, adding new clusters and updating credentials as needed.

Understanding ACP CLI Configuration Structure

ACP CLI stores all configuration information in the standard `~/.kube/config` file, ensuring full compatibility with kubectl and other Kubernetes tools while adding ACP-specific enhancements.

ACP CLI-Enhanced kubeconfig Structure

ACP CLI extends the standard kubeconfig format with ACP-specific metadata for enhanced platform integration:

```
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: <TOKEN>
    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 be
havior, do not edit this item.
```

Metadata Structure and Organization

ACP CLI uses extension metadata to organize and identify configuration entries:

Metadata-Based Identification

- **Platform Identification**: Uses `platformUrl` to identify the parent platform
- **Session Association**: Uses `sessionName` to group related clusters, users, and contexts
- **Global Cluster Detection**: Uses `isGlobal` field to identify management clusters
- **User Credential Location**: Matches `sessionName` and `platformUrl` in user extensions

Naming Conventions

ACP CLI uses consistent naming conventions when creating new entries:

- **Cluster entries**: `acp:<session_name>:<cluster_name>` (e.g., `acp:prod:global`)
- **User entries**: `acp:<session_name>:user` (e.g., `acp:prod:user`)
- **Context entries**: `<session_name>/<cluster_name>` (e.g., `prod/global`)

> **NOTE**
>
> The `acp:` prefix ensures ACP CLI-managed entries don't conflict with existing kubeconfig entries.
>
> Users can manually rename these entries - ACP CLI uses metadata for identification, not names.

Manual Configuration of CLI Profiles

For advanced users who need precise control over configuration, ACP CLI supports all standard kubectl config commands for manual kubeconfig management.

TIP

Most users should use the convenient commands described above.

Manual configuration commands are useful for advanced scenarios:

- **Custom context naming** - Creating contexts that don't follow ACP CLI naming conventions
- **Non-ACP environments** - Managing traditional kubectl contexts alongside ACP sessions
- **Complex multi-context scenarios** - Advanced workflows requiring precise context control
- **Troubleshooting configuration issues** - Debugging or repairing configuration problems

Standard Configuration Commands

ACP CLI provides full compatibility with kubectl config subcommands:

Subcommand	Usage
`set-cluster`	Sets a cluster entry in the CLI config file
`set-context`	Sets a context entry in the CLI config file
`use-context`	Sets the current context using the specified context name
`set`	Sets an individual value in the CLI config file
`unset`	Unsets individual values in the CLI config file
`view`	Displays the merged CLI configuration currently in use

Example Manual Operations

Create a custom context:

```
# Create context with custom naming
ac config set-context my-custom-context --cluster=acp:prod:workload-a --n
amespace=my-app

# Switch to the custom context
ac config use-context my-custom-context
```

View current configuration:

```
# Display merged configuration
ac config view

# Display configuration from specific file
ac config view --config=/path/to/config
```

Update context namespace:

```
# Set namespace for current context
ac config set-context `ac config current-context` --namespace=my-namespac
e
```

Load and Merge Rules

You can follow these rules when issuing CLI operations for the loading and merging order for the CLI configuration:

- CLI config files are retrieved from your workstation, using the following hierarchy and merge rules:
  - If the `--config` option is set, then only that file is loaded. The flag is set once and no merging takes place.
  - If the `$KUBECONFIG` environment variable is set, then it is used. The variable can be a list of paths, and if so the paths are merged together. When a value is modified, it is

modified in the file that defines the stanza. When a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the last file in the list.

- Otherwise, the `~/.kube/config` file is used and no merging takes place.

- The context to use is determined based on the first match in the following flow:
  - The value of the `--context` option.
  - The `current-context` value from the CLI config file.
  - An empty value is allowed at this stage.
- The user and cluster to use is determined. At this point, you may or may not have a context; they are built based on the first match in the following flow, which is run once for the user and once for the cluster:
  - The value of the `--user` for user name and `--cluster` option for cluster name.
  - If the `--context` option is present, then use the context's value.
  - An empty value is allowed at this stage.
- The actual cluster information to use is determined. At this point, you may or may not have cluster information. Each piece of the cluster information is built based on the first match in the following flow:
  - The values of any of the following command-line options: `--server`, `--api-version`, `--certificate-authority`, `--insecure-skip-tls-verify`
  - If cluster information and a value for the attribute is present, then use it.
  - If you do not have a server location, then there is an error.
- The actual user information to use is determined. Users are built using the same rules as clusters, except that you can only have one authentication technique per user; conflicting techniques cause the operation to fail. Command-line options take precedence over config file values. Valid command-line options are:
  - `--auth-path`
  - `--client-certificate`
  - `--client-key`
  - `--token`

- For any information that is still missing, default values are used and prompts are given for additional information.

Extending ACP CLI with Plugins

You can write and install plugins to build on the default ac commands, allowing you to perform new and more complex tasks with ACP CLI and ACP platform integration.

TOC

- Writing CLI Plugins
  - Creating a Simple Plugin
  - Plugin Development Requirements
  - Additional Resources
- Installing and Using CLI Plugins
  - Prerequisites
  - Installation Procedure

Writing CLI Plugins

You can write a plugin for ACP CLI (ac) in any programming language or script that allows you to write command-line commands. Note that you cannot use a plugin to overwrite an existing ac command.

Creating a Simple Plugin

This procedure creates a simple Bash plugin that prints a message to the terminal when the `ac foo` command is issued.

Procedure

1. Create a file called `ac-foo`. When naming your plugin file, keep the following in mind:
   - The file must begin with `ac-` or `kubectl-` to be recognized as a plugin
   - The file name determines the command that invokes the plugin. For example, a plugin with the file name `ac-foo-bar` can be invoked by a command of `ac foo bar`
   - You can also use underscores if you want the command to contain dashes. For example, a plugin with the file name `ac-foo_bar` can be invoked by a command of `ac foo-bar`
2. Add the following contents to the file:

```
#!/bin/bash

# Optional argument handling
if [[ "$1" == "version" ]]; then
    echo "1.0.0"
    exit 0
fi

# Optional argument handling
if [[ "$1" == "config" ]]; then
    echo $KUBECONFIG
    exit 0
fi

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

   After you install this plugin for ACP CLI, it can be invoked using the `ac foo` command.

Plugin Development Requirements

- **Programming Language**: Use any programming language or script that supports command-line interfaces

- **Naming Convention**: Plugin files must follow the `ac-<plugin-name>` or `kubectl-<plugin-name>` naming pattern
- **Executable**: Plugin files must have executable permissions
- **Command Overrides**: Plugins cannot overwrite existing ACP CLI commands
- **Argument Handling**: Plugins should handle standard command-line arguments and flags appropriately

Additional Resources

- Review kubectl plugin development guides for implementation patterns and best practices
- Use CLI runtime utilities for Go-based plugin development
- Consider ACP platform integration when designing plugins that interact with cluster resources

Installing and Using CLI Plugins

After you write a custom plugin for ACP CLI, you must install the plugin before use.

Prerequisites

- You must have ACP CLI (ac) installed
- You must have a CLI plugin file that begins with `ac-` or `kubectl-`

Installation Procedure

1. If necessary, update the plugin file to be executable:

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

2. Place the file anywhere in your PATH, such as `/usr/local/bin/`:

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

3. Run `ac plugin list` to make sure that the plugin is listed:

   ```
   ac plugin list
   ```

   Example output

   ```
   The following compatible plugins are available:

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

   If your plugin is not listed here, verify that the file begins with `ac-` or `kubectl-`, is executable, and is on your PATH.

4. Invoke the new command or option introduced by the plugin.
   For example, if you built and installed the `ac-ns` plugin, you can use the following command to view the current namespace:

   ```
   ac ns
   ```

   Note that the command to invoke the plugin depends on the plugin file name. For example, a plugin with the file name of `ac-foo-bar` is invoked by the `ac foo bar` command.

AC CLI Developer Command Reference

This reference provides descriptions and example commands for AC CLI developer commands. For administrator commands, see the AC CLI administrator command reference.

Run `ac help` to list all commands or run `ac <command> --help` to get additional details for a specific command.

TOC

ac annotate
Example usage
ac api-resources
Example usage
ac api-versions
Example usage
ac apply
Example usage
ac apply edit-last-applied
Example usage
ac apply set-last-applied
Example usage
ac apply view-last-applied
Example usage
ac attach

Example usage
ac auth
ac auth can-i
Example usage
ac auth reconcile
Example usage
ac auth whoami
Example usage
ac autoscale
Example usage
ac cluster-info
Example usage
ac cluster-info dump
Example usage
ac completion
Example usage
ac config
ac config current-context
Example usage
ac config delete-cluster
Example usage
ac config delete-context
Example usage
ac config delete-user
Example usage
ac config get-clusters
Example usage
ac config get-contexts
Example usage
ac config get-sessions
Example usage
ac config get-users

Example usage
ac config rename-context
Example usage
ac config set
Example usage
ac config set-cluster
Example usage
ac config set-context
Example usage
ac config set-credentials
Example usage
ac config sync
Example usage
ac config unset
Example usage
ac config use-cluster
Example usage
ac config use-context
Example usage
ac config use-session
Example usage
ac config view
Example usage
ac cp
Example usage
ac create
Example usage
ac create clusterrole
Example usage
ac create clusterrolebinding
Example usage
ac create configmap

Example usage

ac create cronjob

Example usage

ac create deployment

Example usage

ac create ingress

Example usage

ac create job

Example usage

ac create namespace

Example usage

ac create poddisruptionbudget

Example usage

ac create priorityclass

Example usage

ac create quota

Example usage

ac create role

Example usage

ac create rolebinding

Example usage

ac create secret

ac create secret docker-registry

Example usage

ac create secret generic

Example usage

ac create secret tls

Example usage

ac create service

ac create service clusterip

Example usage

ac create service externalname

Example usage

ac create service loadbalancer

Example usage

ac create service nodeport

Example usage

ac create serviceaccount

Example usage

ac create token

Example usage

ac delete

Example usage

ac describe

Example usage

ac diff

Example usage

ac edit

Example usage

ac events

Example usage

ac exec

Example usage

ac explain

Example usage

ac expose

Example usage

ac get

Example usage

ac kustomize

Example usage

ac label

Example usage

ac login

Example usage
ac logout
Example usage
ac logs
Example usage
ac namespace
Example usage
ac patch
Example usage
ac plugin
Example usage
ac plugin list
Example usage
ac port-forward
Example usage
ac process
Example usage
ac proxy
Example usage
ac replace
Example usage
ac rollout
Example usage
ac rollout history
Example usage
ac rollout pause
Example usage
ac rollout restart
Example usage
ac rollout resume
Example usage
ac rollout status

Example usage
ac rollout undo
Example usage
ac run
Example usage
ac scale
Example usage
ac set
ac set env
Example usage
ac set image
Example usage
ac set resources
Example usage
ac set selector
Example usage
ac set serviceaccount
Example usage
ac set subject
Example usage
ac version
Example usage
ac wait
Example usage
ac whoami
Example usage

ac annotate

Update the annotations on a resource

Example usage

```
# Update pod 'foo' with the annotation 'description' and the value 'my fr
ontend'
# If the same annotation is set multiple times, only the last value will
be applied
ac annotate pods foo description='my frontend'

# Update a pod identified by type and name in "pod.json"
ac annotate -f pod.json description='my frontend'

# Update pod 'foo' with the annotation 'description' and the value 'my fr
ontend running nginx', overwriting any existing value
ac annotate --overwrite pods foo description='my frontend running nginx'

# Update all pods in the namespace
ac annotate pods --all description='my frontend running nginx'

# Update pod 'foo' only if the resource is unchanged from version 1
ac annotate pods foo description='my frontend running nginx' --resource-v
ersion=1

# Update pod 'foo' by removing an annotation named 'description' if it ex
ists
# Does not require the --overwrite flag
ac annotate pods foo description-
```

ac api-resources

Print the supported API resources on the server

Example usage

```
# Print the supported API resources
ac api-resources

# Print the supported API resources with more information
ac api-resources -o wide

# Print the supported API resources sorted by a column
ac api-resources --sort-by=name

# Print the supported namespaced resources
ac api-resources --namespaced=true

# Print the supported non-namespaced resources
ac api-resources --namespaced=false

# Print the supported API resources with a specific APIGroup
ac api-resources --api-group=rbac.authorization.k8s.io
```

ac api-versions

Print the supported API versions on the server, in the form of "group/version"

Example usage

```
# Print the supported API versions
ac api-versions
```

ac apply

Apply a configuration to a resource by file name or stdin

Example usage

```
# Apply the configuration in pod.json to a pod
ac apply -f ./pod.json

# Apply resources from a directory containing kustomization.yaml - e.g. d
ir/kustomization.yaml
ac apply -k dir/

# Apply the JSON passed into stdin to a pod
cat pod.json | ac apply -f -

# Apply the configuration from all files that end with '.json'
ac apply -f '*.json'

# Note: --prune is still in Alpha
# Apply the configuration in manifest.yaml that matches label app=nginx a
nd delete all other resources that are not in the file and match label ap
p=nginx
ac apply --prune -f manifest.yaml -l app=nginx

# Apply the configuration in manifest.yaml and delete all the other confi
g maps that are not in the file
ac apply --prune -f manifest.yaml --all --prune-allowlist=core/v1/ConfigM
ap
```

ac apply edit-last-applied

Edit latest last-applied-configuration annotations of a resource/object

Example usage

```
# Edit the last-applied-configuration annotations by type/name in YAML
ac apply edit-last-applied deployment/nginx

# Edit the last-applied-configuration annotations by file in JSON
ac apply edit-last-applied -f deploy.yaml -o json
```

ac apply set-last-applied

Set the last-applied-configuration annotation on a live object to match the contents of a file

Example usage

```
# Set the last-applied-configuration of a resource to match the contents
of a file
ac apply set-last-applied -f deploy.yaml

# Execute set-last-applied against each configuration file in a directory
ac apply set-last-applied -f path/

# Set the last-applied-configuration of a resource to match the contents
of a file; will create the annotation if it does not already exist
ac apply set-last-applied -f deploy.yaml --create-annotation=true
```

ac apply view-last-applied

View the latest last-applied-configuration annotations of a resource/object

Example usage

```
# View the last-applied-configuration annotations by type/name in YAML
ac apply view-last-applied deployment/nginx

# View the last-applied-configuration annotations by file in JSON
ac apply view-last-applied -f deploy.yaml -o json
```

ac attach

Attach to a running container

Example usage

```
# Get output from running pod mypod; use the 'ac.kubernetes.io/default-co
ntainer' annotation
# for selecting the container to be attached or the first container in th
e pod will be chosen
ac attach mypod

# Get output from ruby-container from pod mypod
ac attach mypod -c ruby-container

# Switch to raw terminal mode; sends stdin to 'bash' in ruby-container fr
om pod mypod
# and sends stdout/stderr from 'bash' back to the client
ac attach mypod -c ruby-container -i -t

# Get output from the first pod of a replica set named nginx
ac attach rs/nginx
```

ac auth

Inspect authorization

ac auth can-i

Check whether an action is allowed

Example usage

```
# Check to see if I can create pods in any namespace
ac auth can-i create pods --all-namespaces

# Check to see if I can list deployments in my current namespace
ac auth can-i list deployments.apps

# Check to see if service account "foo" of namespace "dev" can list pods
in the namespace "prod"
# You must be allowed to use impersonation for the global option "--as"
ac auth can-i list pods --as=system:serviceaccount:dev:foo -n prod

# Check to see if I can do everything in my current namespace ("*" means
all)
ac auth can-i '*' '*'

# Check to see if I can get the job named "bar" in namespace "foo"
ac auth can-i list jobs.batch/bar -n foo

# Check to see if I can read pod logs
ac auth can-i get pods --subresource=log

# Check to see if I can access the URL /logs/
ac auth can-i get /logs/

# Check to see if I can approve certificates.k8s.io
ac auth can-i approve certificates.k8s.io

# List all allowed actions in namespace "foo"
ac auth can-i --list --namespace=foo
```

ac auth reconcile

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects

Example usage

```
# Reconcile RBAC resources from a file
ac auth reconcile -f my-rbac-rules.yaml
```

ac auth whoami

Experimental: Check self subject attributes

Example usage

```
# Get your subject attributes
ac auth whoami

# Get your subject attributes in JSON format
ac auth whoami -o json
```

ac autoscale

Auto-scale a deployment, replica set, stateful set, or replication controller

Example usage

```
# Auto scale a deployment "foo", with the number of pods between 2 and 1
0, no target CPU utilization specified so a default autoscaling policy wi
ll be used
ac autoscale deployment foo --min=2 --max=10

# Auto scale a replication controller "foo", with the number of pods betw
een 1 and 5, target CPU utilization at 80%
ac autoscale rc foo --max=5 --cpu-percent=80
```

ac cluster-info

Display cluster information

Example usage

```
# Print the address of the control plane and cluster services
ac cluster-info
```

ac cluster-info dump

Dump relevant information for debugging and diagnosis

Example usage

```
# Dump current cluster state to stdout
ac cluster-info dump

# Dump current cluster state to /path/to/cluster-state
ac cluster-info dump --output-directory=/path/to/cluster-state

# Dump all namespaces to stdout
ac cluster-info dump --all-namespaces

# Dump a set of namespaces to /path/to/cluster-state
ac cluster-info dump --namespaces default,kube-system --output-directory
=/path/to/cluster-state
```

ac completion

Output shell completion code for the specified shell (bash, zsh, fish, or powershell)

Example usage

```
# Installing bash completion on macOS using homebrew
## If running Bash 3.2 included with macOS
brew install bash-completion
## or, if running Bash 4.1+
brew install bash-completion@2
## If ac is installed via homebrew, this should start working immediately
## If you've installed via other means, you may need add the completion t
o your completion directory
ac completion bash > $(brew --prefix)/etc/bash_completion.d/ac


# Installing bash completion on Linux
## If bash-completion is not installed on Linux, install the 'bash-comple
tion' package
## via your distribution's package manager.
## Load the ac completion code for bash into the current shell
source <(ac completion bash)
## Write bash completion code to a file and source it from .bash_profile
ac completion bash > ~/.kube/completion.bash.inc
printf "
# ac shell completion
source '$HOME/.kube/completion.bash.inc'
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# Load the ac completion code for zsh[1] into the current shell
source <(ac completion zsh)
# Set the ac completion code for zsh[1] to autoload on startup
ac completion zsh > "${fpath[1]}/_ac"


# Load the ac completion code for fish[2] into the current shell
ac completion fish | source
# To load completions for each session, execute once:
ac completion fish > ~/.config/fish/completions/ac.fish

# Load the ac completion code for powershell into the current shell
ac completion powershell | Out-String | Invoke-Expression
# Set ac completion code for powershell to run on startup
## Save completion code to a script and execute in the profile
ac completion powershell > $HOME\.kube\completion.ps1
Add-Content $PROFILE "$HOME\.kube\completion.ps1"
## Execute completion code in the profile
```

```
Add-Content $PROFILE "if (Get-Command ac -ErrorAction SilentlyContinue) {
ac completion powershell | Out-String | Invoke-Expression
}"
## Add completion code directly to the $PROFILE script
ac completion powershell >> $PROFILE
```

ac config

Modify kubeconfig files

ac config current-context

Display the current-context

Example usage

```
# Display the current-context
ac config current-context
```

ac config delete-cluster

Delete the specified cluster from the kubeconfig

Example usage

```
# Delete the minikube cluster
ac config delete-cluster minikube
```

ac config delete-context

Delete the specified context from the kubeconfig

Example usage

```
# Delete the context for the minikube cluster
ac config delete-context minikube
```

ac config delete-user

Delete the specified user from the kubeconfig

Example usage

```
# Delete the minikube user
ac config delete-user minikube
```

ac config get-clusters

Display clusters defined in the kubeconfig

Example usage

```
# List the clusters that ac knows about
ac config get-clusters
```

ac config get-contexts

Describe one or many contexts

Example usage

```
# List all the contexts in your kubeconfig file
ac config get-contexts

# Describe one context in your kubeconfig file
ac config get-contexts my-context
```

ac config get-sessions

List all configured ACP platform sessions

Example usage

```
# List all configured ACP sessions
ac config get-sessions
```

ac config get-users

Display users defined in the kubeconfig

Example usage

```
# List the users that ac knows about
ac config get-users
```

ac config rename-context

Rename a context from the kubeconfig file

Example usage

```
# Rename the context 'old-name' to 'new-name' in your kubeconfig file
ac config rename-context old-name new-name
```

ac config set

Set an individual value in a kubeconfig file

Example usage

```
# Set the server field on the my-cluster cluster to https://1.2.3.4
ac config set clusters.my-cluster.server https://1.2.3.4

# Set the certificate-authority-data field on the my-cluster cluster
ac config set clusters.my-cluster.certificate-authority-data $(echo "cert
_data_here" | base64 -i -)

# Set the cluster field in the my-context context to my-cluster
ac config set contexts.my-context.cluster my-cluster

# Set the client-key-data field in the cluster-admin user using --set-raw
-bytes option
ac config set users.cluster-admin.client-key-data cert_data_here --set-ra
w-bytes=true
```

ac config set-cluster

Set a cluster entry in kubeconfig

Example usage

```
# Set only the server field on the e2e cluster entry without touching oth
er values
ac config set-cluster e2e --server=https://1.2.3.4

# Embed certificate authority data for the e2e cluster entry
ac config set-cluster e2e --embed-certs --certificate-authority=~/.kube/e
2e/kubernetes.ca.crt

# Disable cert checking for the e2e cluster entry
ac config set-cluster e2e --insecure-skip-tls-verify=true

# Set the custom TLS server name to use for validation for the e2e cluste
r entry
ac config set-cluster e2e --tls-server-name=my-cluster-name

# Set the proxy URL for the e2e cluster entry
ac config set-cluster e2e --proxy-url=https://1.2.3.4
```

ac config set-context

Set a context entry in kubeconfig

Example usage

```
# Set the user field on the gce context entry without touching other valu
es
ac config set-context gce --user=cluster-admin
```

ac config set-credentials

Set a user entry in kubeconfig

Example usage

```
# Set only the "client-key" field on the "cluster-admin"
# entry, without touching other values
ac config set-credentials cluster-admin --client-key=~/.kube/admin.key

# Set basic auth for the "cluster-admin" entry
ac config set-credentials cluster-admin --username=admin --password=uXFGw
eU9l35qcif

# Embed client certificate data in the "cluster-admin" entry
ac config set-credentials cluster-admin --client-certificate=~/.kube/admi
n.crt --embed-certs=true

# Enable the Google Compute Platform auth provider for the "cluster-admi
n" entry
ac config set-credentials cluster-admin --auth-provider=gcp

# Enable the OpenID Connect auth provider for the "cluster-admin" entry w
ith additional arguments
ac config set-credentials cluster-admin --auth-provider=oidc --auth-provi
der-arg=client-id=foo --auth-provider-arg=client-secret=bar

# Remove the "client-secret" config value for the OpenID Connect auth pro
vider for the "cluster-admin" entry
ac config set-credentials cluster-admin --auth-provider=oidc --auth-provi
der-arg=client-secret-

# Enable new exec auth plugin for the "cluster-admin" entry
ac config set-credentials cluster-admin --exec-command=/path/to/the/execu
table --exec-api-version=client.authentication.k8s.io/v1beta1

# Enable new exec auth plugin for the "cluster-admin" entry with interact
ive mode
ac config set-credentials cluster-admin --exec-command=/path/to/the/execu
table --exec-api-version=client.authentication.k8s.io/v1beta1 --exec-inte
ractive-mode=Never

# Define new exec auth plugin arguments for the "cluster-admin" entry
ac config set-credentials cluster-admin --exec-arg=arg1 --exec-arg=arg2

# Create or update exec auth plugin environment variables for the "cluste
r-admin" entry
ac config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=k
ey2=val2
```

```
# Remove exec auth plugin environment variables for the "cluster-admin" e
ntry
ac config set-credentials cluster-admin --exec-env=var-to-remove-
```

ac config sync

Synchronize kubeconfig with ACP platform state

Example usage

```
# Sync current session based on active context
ac config sync

# Sync specific session
ac config sync --session prod

# Sync all sessions
ac config sync --all
```

ac config unset

Unset an individual value in a kubeconfig file

Example usage

```
# Unset the current-context
ac config unset current-context

# Unset namespace in foo context
ac config unset contexts.foo.namespace
```

ac config use-cluster

Switch to a specific ACP cluster by cluster name

Example usage

```
# Switch to existing context for workload-a cluster
ac config use-cluster workload-a

# Create new context for workload-b with namespace
ac config use-cluster workload-b --namespace my-app

# Switch to global cluster
ac config use-cluster global
```

ac config use-context

Set the current-context in a kubeconfig file

Example usage

```
# Use the context for the minikube cluster
ac config use-context minikube
```

ac config use-session

Switch to a specified ACP session with intelligent context selection

Example usage

```
# Switch to staging session (default global cluster)
ac config use-session staging

# Switch to production session with specific cluster
ac config use-session prod --cluster workload-b

# Switch to session with specific cluster and namespace
ac config use-session staging --cluster workload-a --namespace my-app
```

ac config view

Display merged kubeconfig settings or a specified kubeconfig file

Example usage

```
# Show merged kubeconfig settings
ac config view

# Show merged kubeconfig settings, raw certificate data, and exposed secr
ets
ac config view --raw

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

ac cp

Copy files and directories to and from containers

Example usage

```
# !!!Important Note!!!
# Requires that the 'tar' binary is present in your container
# image.  If 'tar' is not present, 'ac cp' will fail.
#
# For advanced use cases, such as symlinks, wildcard expansion or
# file mode preservation, consider using 'ac exec'.

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some
-namespace>
tar cf - /tmp/foo | ac exec -i -n <some-namespace> <some-pod> -- tar xf -
-C /tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
ac exec -n <some-namespace> <some-pod> -- tar cf - /tmp/foo | tar xf - -C
/tmp/bar

# Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod in th
e default namespace
ac cp /tmp/foo_dir <some-pod>:/tmp/bar_dir

# Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific cont
ainer
ac cp /tmp/foo <some-pod>:/tmp/bar -c <specific-container>

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some
-namespace>
ac cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
ac cp <some-namespace>/<some-pod>:/tmp/foo /tmp/bar
```

ac create

Create a resource from a file or from stdin

Example usage

```
# Create a pod using the data in pod.json
ac create -f ./pod.json

# Create a pod based on the JSON passed into stdin
cat pod.json | ac create -f -

# Edit the data in registry.yaml in JSON then create the resource using t
he edited data
ac create -f registry.yaml --edit -o json
```

ac create clusterrole

Create a cluster role

Example usage

```
# Create a cluster role named "pod-reader" that allows user to perform "g
et", "watch" and "list" on pods
ac create clusterrole pod-reader --verb=get,list,watch --resource=pods

# Create a cluster role named "pod-reader" with ResourceName specified
ac create clusterrole pod-reader --verb=get --resource=pods --resource-na
me=readablepod --resource-name=anotherpod

# Create a cluster role named "foo" with API Group specified
ac create clusterrole foo --verb=get,list,watch --resource=rs.apps

# Create a cluster role named "foo" with SubResource specified
ac create clusterrole foo --verb=get,list,watch --resource=pods,pods/stat
us

# Create a cluster role name "foo" with NonResourceURL specified
ac create clusterrole "foo" --verb=get --non-resource-url=/logs/*

# Create a cluster role name "monitoring" with AggregationRule specified
ac create clusterrole monitoring --aggregation-rule="rbac.example.com/agg
regate-to-monitoring=true"
```

ac create clusterrolebinding

Create a cluster role binding for a particular cluster role

Example usage

```
# Create a cluster role binding for user1, user2, and group1 using the cl
uster-admin cluster role
ac create clusterrolebinding cluster-admin --clusterrole=cluster-admin --
user=user1 --user=user2 --group=group1
```

ac create configmap

Create a config map from a local file, directory or literal value

Example usage

```
# Create a new config map named my-config based on folder bar
ac create configmap my-config --from-file=path/to/bar

# Create a new config map named my-config with specified keys instead of
file basenames on disk
ac create configmap my-config --from-file=key1=/path/to/bar/file1.txt --f
rom-file=key2=/path/to/bar/file2.txt

# Create a new config map named my-config with key1=config1 and key2=conf
ig2
ac create configmap my-config --from-literal=key1=config1 --from-literal=
key2=config2

# Create a new config map named my-config from the key=value pairs in the
file
ac create configmap my-config --from-file=path/to/bar

# Create a new config map named my-config from an env file
ac create configmap my-config --from-env-file=path/to/foo.env --from-env-
file=path/to/bar.env
```

ac create cronjob

Create a cron job with the specified name

Example usage

```
# Create a cron job
ac create cronjob my-job --image=busybox --schedule="*/1 * * * *"

# Create a cron job with a command
ac create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

ac create deployment

Create a deployment with the specified name

Example usage

```
# Create a deployment named my-dep that runs the busybox image
ac create deployment my-dep --image=busybox

# Create a deployment with a command
ac create deployment my-dep --image=busybox -- date

# Create a deployment named my-dep that runs the nginx image with 3 repli
cas
ac create deployment my-dep --image=nginx --replicas=3

# Create a deployment named my-dep that runs the busybox image and expose
port 5701
ac create deployment my-dep --image=busybox --port=5701

# Create a deployment named my-dep that runs multiple containers
ac create deployment my-dep --image=busybox:latest --image=ubuntu:latest
--image=nginx
```

ac create ingress

Create an ingress with the specified name

Example usage

```
# Create a single ingress called 'simple' that directs requests to foo.co
m/bar to svc
# svc1:8080 with a TLS secret "my-cert"
ac create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

# Create a catch all ingress of "/path" pointing to service svc:port and
Ingress Class as "otheringress"
ac create ingress catch-all --class=otheringress --rule="/path=svc:port"

# Create an ingress with two annotations: ingress.annotation1 and ingres
s.annotations2
ac create ingress annotated --class=default --rule="foo.com/bar=svc:port"
\
--annotation ingress.annotation1=foo \
--annotation ingress.annotation2=bla

# Create an ingress with the same host and multiple paths
ac create ingress multipath --class=default \
--rule="foo.com/=svc:port" \
--rule="foo.com/admin/=svcadmin:portadmin"

# Create an ingress with multiple hosts and the pathType as Prefix
ac create ingress ingress1 --class=default \
--rule="foo.com/path*=svc:8080" \
--rule="bar.com/admin*=svc2:http"

# Create an ingress with TLS enabled using the default ingress certificat
e and different path types
ac create ingress ingtls --class=default \
--rule="foo.com/=svc:https,tls" \
--rule="foo.com/path/subpath*=othersvc:8080"

# Create an ingress with TLS enabled using a specific secret and pathType
as Prefix
ac create ingress ingsecret --class=default \
--rule="foo.com/*=svc:8080,tls=secret1"

# Create an ingress with a default backend
ac create ingress ingdefault --class=default \
--default-backend=defaultsvc:http \
--rule="foo.com/*=svc:8080,tls=secret1"
```

ac create job

Create a job with the specified name

Example usage

```
# Create a job
ac create job my-job --image=busybox

# Create a job with a command
ac create job my-job --image=busybox -- date

# Create a job from a cron job named "a-cronjob"
ac create job test-job --from=cronjob/a-cronjob
```

ac create namespace

Create a namespace with the specified name

Example usage

```
# Create a new namespace named my-namespace
ac create namespace my-namespace
```

ac create poddisruptionbudget

Create a pod disruption budget with the specified name

Example usage

```
# Create a pod disruption budget named my-pdb that will select all pods w
ith the app=rails label
# and require at least one of them being available at any point in time
ac create poddisruptionbudget my-pdb --selector=app=rails --min-available
=1

# Create a pod disruption budget named my-pdb that will select all pods w
ith the app=nginx label
# and require at least half of the pods selected to be available at any p
oint in time
ac create pdb my-pdb --selector=app=nginx --min-available=50%
```

ac create priorityclass

Create a priority class with the specified name

Example usage

```
# Create a priority class named high-priority
ac create priorityclass high-priority --value=1000 --description="high pr
iority"

# Create a priority class named default-priority that is considered as th
e global default priority
ac create priorityclass default-priority --value=1000 --global-default=tr
ue --description="default priority"

# Create a priority class named high-priority that cannot preempt pods wi
th lower priority
ac create priorityclass high-priority --value=1000 --description="high pr
iority" --preemption-policy="Never"
```

ac create quota

Create a quota with the specified name

Example usage

```
# Create a new resource quota named my-quota
ac create quota my-quota --hard=cpu=1,memory=1G,pods=2,services=3,replica
tioncontrollers=2,resourcequotas=1,secrets=5,persistentvolumeclaims=10

# Create a new resource quota named best-effort
ac create quota best-effort --hard=pods=100 --scopes=BestEffort
```

ac create role

Create a role with single rule

Example usage

```
# Create a role named "pod-reader" that allows user to perform "get", "wa
tch" and "list" on pods
ac create role pod-reader --verb=get --verb=list --verb=watch --resource=
pods

# Create a role named "pod-reader" with ResourceName specified
ac create role pod-reader --verb=get --resource=pods --resource-name=read
ablepod --resource-name=anotherpod

# Create a role named "foo" with API Group specified
ac create role foo --verb=get,list,watch --resource=rs.apps

# Create a role named "foo" with SubResource specified
ac create role foo --verb=get,list,watch --resource=pods,pods/status
```

ac create rolebinding

Create a role binding for a particular role or cluster role

Example usage

```
# Create a role binding for user1, user2, and group1 using the admin clus
ter role
ac create rolebinding admin --clusterrole=admin --user=user1 --user=user2
--group=group1

# Create a role binding for service account monitoring:sa-dev using the a
dmin role
ac create rolebinding admin-binding --role=admin --serviceaccount=monitor
ing:sa-dev
```

ac create secret

Create a secret using a specified subcommand

ac create secret docker-registry

Create a secret for use with a Docker registry

Example usage

```
# If you do not already have a .dockercfg file, create a dockercfg secret
directly
ac create secret docker-registry my-secret --docker-server=DOCKER_REGISTR
Y_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD
--docker-email=DOCKER_EMAIL

# Create a new secret named my-secret from ~/.docker/config.json
ac create secret docker-registry my-secret --from-file=path/to/.docker/co
nfig.json
```

ac create secret generic

Create a secret from a local file, directory, or literal value

Example usage

```
# Create a new secret named my-secret with keys for each file in folder b
ar
ac create secret generic my-secret --from-file=path/to/bar

# Create a new secret named my-secret with specified keys instead of name
s on disk
ac create secret generic my-secret --from-file=ssh-privatekey=path/to/id_
rsa --from-file=ssh-publickey=path/to/id_rsa.pub

# Create a new secret named my-secret with key1=supersecret and key2=tops
ecret
ac create secret generic my-secret --from-literal=key1=supersecret --from
-literal=key2=topsecret

# Create a new secret named my-secret using a combination of a file and a
literal
ac create secret generic my-secret --from-file=ssh-privatekey=path/to/id_
rsa --from-literal=passphrase=topsecret

# Create a new secret named my-secret from env files
ac create secret generic my-secret --from-env-file=path/to/foo.env --from
-env-file=path/to/bar.env
```

ac create secret tls

Create a TLS secret

Example usage

```
# Create a new TLS secret named tls-secret with the given key pair
ac create secret tls tls-secret --cert=path/to/tls.crt --key=path/to/tls.
key
```

ac create service

Create a service using a specified subcommand

ac create service clusterip

Create a ClusterIP service

Example usage

```
# Create a new ClusterIP service named my-cs
ac create service clusterip my-cs --tcp=5678:8080

# Create a new ClusterIP service named my-cs (in headless mode)
ac create service clusterip my-cs --clusterip="None"
```

ac create service externalname

Create an ExternalName service

Example usage

```
# Create a new ExternalName service named my-ns
ac create service externalname my-ns --external-name bar.com
```

ac create service loadbalancer

Create a LoadBalancer service

Example usage

```
# Create a new LoadBalancer service named my-lbs
ac create service loadbalancer my-lbs --tcp=5678:8080
```

ac create service nodeport

Create a NodePort service

Example usage

```
# Create a new NodePort service named my-ns
ac create service nodeport my-ns --tcp=5678:8080
```

ac create serviceaccount

Create a service account with the specified name

Example usage

```
# Create a new service account named my-service-account
ac create serviceaccount my-service-account
```

ac create token

Request a service account token

Example usage

```
# Request a token to authenticate to the kube-apiserver as the service ac
count "myapp" in the current namespace
ac create token myapp

# Request a token for a service account in a custom namespace
ac create token myapp --namespace myns

# Request a token with a custom expiration
ac create token myapp --duration 10m

# Request a token with a custom audience
ac create token myapp --audience https://example.com

# Request a token bound to an instance of a Secret object
ac create token myapp --bound-object-kind Secret --bound-object-name myse
cret

# Request a token bound to an instance of a Secret object with a specific
UID
ac create token myapp --bound-object-kind Secret --bound-object-name myse
cret --bound-object-uid 0d4691ed-659b-4935-a832-355f77ee47cc
```

ac delete

Delete resources by file names, stdin, resources and names, or by resources and label selector

Example usage

```
# Delete a pod using the type and name specified in pod.json
ac delete -f ./pod.json

# Delete resources from a directory containing kustomization.yaml - e.g.
dir/kustomization.yaml
ac delete -k dir

# Delete resources from all files that end with '.json'
ac delete -f '*.json'

# Delete a pod based on the type and name in the JSON passed into stdin
cat pod.json | ac delete -f -

# Delete pods and services with same names "baz" and "foo"
ac delete pod,service baz foo

# Delete pods and services with label name=myLabel
ac delete pods,services -l name=myLabel

# Delete a pod with minimal delay
ac delete pod foo --now

# Force delete a pod on a dead node
ac delete pod foo --force

# Delete all pods
ac delete pods --all

# Delete all pods only if the user confirms the deletion
ac delete pods --all --interactive
```

ac describe

Show details of a specific resource or group of resources

Example usage

```
# Describe a node
ac describe nodes kubernetes-node-emt8.c.myproject.internal

# Describe a pod
ac describe pods/nginx

# Describe a pod identified by type and name in "pod.json"
ac describe -f pod.json

# Describe all pods
ac describe pods

# Describe pods by label name=myLabel
ac describe pods -l name=myLabel

# Describe all pods managed by the 'frontend' replication controller
# (rc-created pods get the name of the rc as a prefix in the pod name)
ac describe pods frontend
```

ac diff

Diff the live version against a would-be applied version

Example usage

```
# Diff resources included in pod.json
ac diff -f pod.json

# Diff file read from stdin
cat service.yaml | ac diff -f -
```

ac edit

Edit a resource on the server

Example usage

```
# Edit the service named 'registry'
ac edit svc/registry

# Use an alternative editor
KUBE_EDITOR="nano" ac edit svc/registry

# Edit the job 'myjob' in JSON using the v1 API format
ac edit job.v1.batch/myjob -o json

# Edit the deployment 'mydeployment' in YAML and save the modified config
in its annotation
ac edit deployment/mydeployment -o yaml --save-config

# Edit the 'status' subresource for the 'mydeployment' deployment
ac edit deployment mydeployment --subresource='status'
```

ac events

List events

Example usage

```
# List recent events in the default namespace
ac events

# List recent events in all namespaces
ac events --all-namespaces

# List recent events for the specified pod, then wait for more events and
list them as they arrive
ac events --for pod/web-pod-13je7 --watch

# List recent events in YAML format
ac events -oyaml

# List recent only events of type 'Warning' or 'Normal'
ac events --types=Warning,Normal
```

ac exec

Execute a command in a container

Example usage

```
# Get output from running the 'date' command from pod mypod, using the fi
rst container by default
ac exec mypod -- date

# Get output from running the 'date' command in ruby-container from pod m
ypod
ac exec mypod -c ruby-container -- date

# Switch to raw terminal mode; sends stdin to 'bash' in ruby-container fr
om pod mypod
# and sends stdout/stderr from 'bash' back to the client
ac exec mypod -c ruby-container -i -t -- bash -il

# List contents of /usr from the first container of pod mypod and sort by
modification time
# If the command you want to execute in the pod has any flags in common
(e.g. -i),
# you must use two dashes (--) to separate your command's flags/arguments
# Also note, do not surround your command and its flags/arguments with qu
otes
# unless that is how you would execute it normally (i.e., do ls -t /usr,
not "ls -t /usr")
ac exec mypod -i -t -- ls -t /usr

# Get output from running 'date' command from the first pod of the deploy
ment mydeployment, using the first container by default
ac exec deploy/mydeployment -- date

# Get output from running 'date' command from the first pod of the servic
e myservice, using the first container by default
ac exec svc/myservice -- date
```

ac explain

Get documentation for a resource

Example usage

```
# Get the documentation of the resource and its fields
ac explain pods

# Get all the fields in the resource
ac explain pods --recursive

# Get the explanation for deployment in supported api versions
ac explain deployments --api-version=apps/v1

# Get the documentation of a specific field of a resource
ac explain pods.spec.containers

# Get the documentation of resources in different format
ac explain deployment --output=plaintext-openapiv2
```

ac expose

Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service

Example usage

```
# Create a service for a replicated nginx, which serves on port 80 and co
nnects to the containers on port 8000
ac expose rc nginx --port=80 --target-port=8000

# Create a service for a replication controller identified by type and na
me specified in "nginx-controller.yaml", which serves on port 80 and conn
ects to the containers on port 8000
ac expose -f nginx-controller.yaml --port=80 --target-port=8000

# Create a service for a pod valid-pod, which serves on port 444 with the
name "frontend"
ac expose pod valid-pod --port=444 --name=frontend

# Create a second service based on the above service, exposing the contai
ner port 8443 as port 443 with the name "nginx-https"
ac expose service nginx --port=443 --target-port=8443 --name=nginx-https

# Create a service for a replicated streaming application on port 4100 ba
lancing UDP traffic and named 'video-stream'.
ac expose rc streamer --port=4100 --protocol=UDP --name=video-stream

# Create a service for a replicated nginx using replica set, which serves
on port 80 and connects to the containers on port 8000
ac expose rs nginx --port=80 --target-port=8000

# Create a service for an nginx deployment, which serves on port 80 and c
onnects to the containers on port 8000
ac expose deployment nginx --port=80 --target-port=8000
```

ac get

Display one or many resources

Example usage

```
# List all pods in ps output format
ac get pods

# List all pods in ps output format with more information (such as node n
ame)
ac get pods -o wide

# List a single replication controller with specified NAME in ps output f
ormat
ac get replicationcontroller web

# List deployments in JSON output format, in the "v1" version of the "app
s" API group
ac get deployments.v1.apps -o json

# List a single pod in JSON output format
ac get -o json pod web-pod-13je7

# List a pod identified by type and name specified in "pod.yaml" in JSON
output format
ac get -f pod.yaml -o json

# List resources from a directory with kustomization.yaml - e.g. dir/kust
omization.yaml
ac get -k dir/

# Return only the phase value of the specified pod
ac get -o template pod/web-pod-13je7 --template={{.status.phase}}

# List resource information in custom columns
ac get pod test-pod -o custom-columns=CONTAINER:.spec.containers[0].name,
IMAGE:.spec.containers[0].image

# List all replication controllers and services together in ps output for
mat
ac get rc,services

# List one or more resources by their type and names
ac get rc/web service/frontend pods/web-pod-13je7

# List the 'status' subresource for a single pod
ac get pod web-pod-13je7 --subresource status
```

```
# List all deployments in namespace 'backend'
ac get deployments.apps --namespace backend

# List all pods existing in all namespaces
ac get pods --all-namespaces
```

ac kustomize

Build a kustomization target from a directory or URL

Example usage

```
# Build the current working directory
ac kustomize

# Build some shared configuration directory
ac kustomize /home/config/production

# Build from github
ac kustomize https://github.com/kubernetes-sigs/kustomize.git/examples/he
lloWorld?ref=v1.0.6
```

ac label

Update the labels on a resource

Example usage

```
# Update pod 'foo' with the label 'unhealthy' and the value 'true'
ac label pods foo unhealthy=true

# Update pod 'foo' with the label 'status' and the value 'unhealthy', ove
rwriting any existing value
ac label --overwrite pods foo status=unhealthy

# Update all pods in the namespace
ac label pods --all status=unhealthy

# Update a pod identified by the type and name in "pod.json"
ac label -f pod.json status=unhealthy

# Update pod 'foo' only if the resource is unchanged from version 1
ac label pods foo status=unhealthy --resource-version=1

# Update pod 'foo' by removing a label named 'bar' if it exists
# Does not require the --overwrite flag
ac label pods foo bar-
```

ac login

Log in to an ACP platform

Example usage

```
# Interactive login (prompts for missing parameters)
ac login https://example.com --name prod

# Login with all parameters via flags
ac login https://example.com --name prod --username=myuser --password=myp
assword

# Login using environment variables (automation-friendly)
AC_LOGIN_PLATFORM_URL=https://example.com AC_LOGIN_SESSION=prod \
AC_LOGIN_USERNAME=myuser AC_LOGIN_PASSWORD=mypassword ac login

# Login with specific identity provider
ac login https://example.com --name prod --idp ldap-test

# Login and set specific cluster and namespace
ac login https://example.com --name prod --cluster=my-cluster --namespace
=my-namespace

# Login with custom kubeconfig file
ac login https://example.com --name prod --kubeconfig=/path/to/kubeconfig

# Login with workload cluster LDAP identity provider
ac login https://192.168.1.2:11780 --idp ldap-test --workload --auth-type
ldap --username 'xx' --password 'xx'

# Login with workload cluster OIDC identity provider
ac login https://192.168.1.2:11780 --idp oidc --workload --auth-type oidc
```

ac logout

End current session with ACP platform

Example usage

```
# Log out from current ACP platform session
ac logout

# Log out from specific session
ac logout --session prod

# Log out from all sessions
ac logout --all
```

ac logs

Print the logs for a container in a pod

Example usage

```
# Return snapshot logs from pod nginx with only one container
ac logs nginx

# Return snapshot logs from pod nginx, prefixing each line with the sourc
e pod and container name
ac logs nginx --prefix

# Return snapshot logs from pod nginx, limiting output to 500 bytes
ac logs nginx --limit-bytes=500

# Return snapshot logs from pod nginx, waiting up to 20 seconds for it to
start running.
ac logs nginx --pod-running-timeout=20s

# Return snapshot logs from pod nginx with multi containers
ac logs nginx --all-containers=true

# Return snapshot logs from all pods in the deployment nginx
ac logs deployment/nginx --all-pods=true

# Return snapshot logs from all containers in pods defined by label app=n
ginx
ac logs -l app=nginx --all-containers=true

# Return snapshot logs from all pods defined by label app=nginx, limiting
concurrent log requests to 10 pods
ac logs -l app=nginx --max-log-requests=10

# Return snapshot of previous terminated ruby container logs from pod web
-1
ac logs -p -c ruby web-1

# Begin streaming the logs from pod nginx, continuing even if errors occu
r
ac logs nginx -f --ignore-errors=true

# Begin streaming the logs of the ruby container in pod web-1
ac logs -f -c ruby web-1

# Begin streaming the logs from all containers in pods defined by label a
pp=nginx
ac logs -f -l app=nginx --all-containers=true
```

```
# Display only the most recent 20 lines of output in pod nginx
ac logs --tail=20 nginx

# Show all logs from pod nginx written in the last hour
ac logs --since=1h nginx

# Show all logs with timestamps from pod nginx starting from August 30, 2
024, at 06:00:00 UTC
ac logs nginx --since-time=2024-08-30T06:00:00Z --timestamps=true

# Show logs from a kubelet with an expired serving certificate
ac logs --insecure-skip-tls-verify-backend nginx

# Return snapshot logs from first container of a job named hello
ac logs job/hello

# Return snapshot logs from container nginx-1 of a deployment named nginx
ac logs deployment/nginx -c nginx-1
```

ac namespace

Show or switch current namespace context

Example usage

```
# Show current namespace and context information
ac namespace

# Switch to a different namespace
ac namespace my-namespace

# Switch to default namespace
ac namespace default
```

ac patch

Update fields of a resource

Example usage

```
# Partially update a node using a strategic merge patch, specifying the p
atch as JSON
ac patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# Partially update a node using a strategic merge patch, specifying the p
atch as YAML
ac patch node k8s-node-1 -p $'spec:\n unschedulable: true'

# Partially update a node identified by the type and name specified in "n
ode.json" using strategic merge patch
ac patch -f node.json -p '{"spec":{"unschedulable":true}}'

# Update a container's image; spec.containers[*].name is required because
it's a merge key
ac patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-ser
ve-hostname","image":"new image"}]}}'

# Update a container's image using a JSON patch with positional arrays
ac patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spe
c/containers/0/image", "value":"new image"}]'

# Update a deployment's replicas through the 'scale' subresource using a
merge patch
ac patch deployment nginx-deployment --subresource='scale' --type='merge'
-p '{"spec":{"replicas":2}}'
```

ac plugin

Provides utilities for interacting with plugins

Example usage

```
# List all available plugins
ac plugin list

# List only binary names of available plugins without paths
ac plugin list --name-only
```

ac plugin list

List all visible plugin executables on a user's PATH

Example usage

```
# List all available plugins
ac plugin list

# List only binary names of available plugins without paths
ac plugin list --name-only
```

ac port-forward

Forward one or more local ports to a pod

Example usage

```
# Listen on ports 5000 and 6000 locally, forwarding data to/from ports 50
00 and 6000 in the pod
ac port-forward pod/mypod 5000 6000

# Listen on ports 5000 and 6000 locally, forwarding data to/from ports 50
00 and 6000 in a pod selected by the deployment
ac port-forward deployment/mydeployment 5000 6000

# Listen on port 8443 locally, forwarding to the targetPort of the servic
e's port named "https" in a pod selected by the service
ac port-forward service/myservice 8443:https

# Listen on port 8888 locally, forwarding to 5000 in the pod
ac port-forward pod/mypod 8888:5000

# Listen on port 8888 on all addresses, forwarding to 5000 in the pod
ac port-forward --address 0.0.0.0 pod/mypod 8888:5000

# Listen on port 8888 on localhost and selected IP, forwarding to 5000 in
the pod
ac port-forward --address localhost,10.19.21.23 pod/mypod 8888:5000

# Listen on a random port locally, forwarding to 5000 in the pod
ac port-forward pod/mypod :5000
```

ac process

Process a template into list of resources

Example usage

```
# Convert the template.json file into a resource list and pass to create
ac process -f template.json | ac apply -f -

# Process a file locally instead of contacting the server
ac process -f template.json -o yaml

# Process template while passing a user-defined label
ac process -f template.json -l name=mytemplate

# Convert a stored template into a resource list
ac process foo

# Convert a stored template into a resource list by setting/overriding pa
rameter values
ac process foo -p PARM1=VALUE1 -p PARM2=VALUE2

# Convert a template stored in different namespace into a resource list
ac process cpaas-system//foo

# Convert template.json into a resource list
cat template.json | ac process -f -
```

ac proxy

Run a proxy to the Kubernetes API server

Example usage

```
# To proxy all of the Kubernetes API and nothing else
ac proxy --api-prefix=/

# To proxy only part of the Kubernetes API and also some static files
# You can get pods info with 'curl localhost:8001/api/v1/pods'
ac proxy --www=/my/files --www-prefix=/static/ --api-prefix=/api/

# To proxy the entire Kubernetes API at a different root
# You can get pods info with 'curl localhost:8001/custom/api/v1/pods'
ac proxy --api-prefix=/custom/

# Run a proxy to the Kubernetes API server on port 8011, serving static c
ontent from ./local/www/
ac proxy --port=8011 --www=./local/www/

# Run a proxy to the Kubernetes API server on an arbitrary local port
# The chosen port for the server will be output to stdout
ac proxy --port=0

# Run a proxy to the Kubernetes API server, changing the API prefix to k8
s-api
# This makes e.g. the pods API available at localhost:8001/k8s-api/v1/pod
s/
ac proxy --api-prefix=/k8s-api
```

ac replace

Replace a resource by file name or stdin

Example usage

```
# Replace a pod using the data in pod.json
ac replace -f ./pod.json

# Replace a pod based on the JSON passed into stdin
cat pod.json | ac replace -f -

# Update a single-container pod's image version (tag) to v4
ac get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | ac rep
lace -f -

# Force replace, delete and then re-create the resource
ac replace --force -f ./pod.json
```

ac rollout

Manage the rollout of a resource

Example usage

```
# Rollback to the previous deployment
ac rollout undo deployment/abc

# Check the rollout status of a daemonset
ac rollout status daemonset/foo

# Restart a deployment
ac rollout restart deployment/abc

# Restart deployments with the 'app=nginx' label
ac rollout restart deployment --selector=app=nginx
```

ac rollout history

View rollout history

Example usage

```
# View the rollout history of a deployment
ac rollout history deployment/abc

# View the details of daemonset revision 3
ac rollout history daemonset/abc --revision=3
```

ac rollout pause

Mark the provided resource as paused

Example usage

```
# Mark the nginx deployment as paused
# Any current state of the deployment will continue its function; new upd
ates
# to the deployment will not have an effect as long as the deployment is
paused
ac rollout pause deployment/nginx
```

ac rollout restart

Restart a resource

Example usage

```
# Restart all deployments in the test-namespace namespace
ac rollout restart deployment -n test-namespace

# Restart a deployment
ac rollout restart deployment/nginx

# Restart a daemon set
ac rollout restart daemonset/abc

# Restart deployments with the app=nginx label
ac rollout restart deployment --selector=app=nginx
```

ac rollout resume

Resume a paused resource

Example usage

```
# Resume an already paused deployment
ac rollout resume deployment/nginx
```

ac rollout status

Show the status of the rollout

Example usage

```
# Watch the rollout status of a deployment
ac rollout status deployment/nginx
```

ac rollout undo

Undo a previous rollout

Example usage

```
# Roll back to the previous deployment
ac rollout undo deployment/abc

# Roll back to daemonset revision 3
ac rollout undo daemonset/abc --to-revision=3

# Roll back to the previous deployment with dry-run
ac rollout undo --dry-run=server deployment/abc
```

ac run

Run a particular image on the cluster

Example usage

```
# Start a nginx pod
ac run nginx --image=nginx

# Start a hazelcast pod and let the container expose port 5701
ac run hazelcast --image=hazelcast/hazelcast --port=5701

# Start a hazelcast pod and set environment variables "DNS_DOMAIN=cluste
r" and "POD_NAMESPACE=default" in the container
ac run hazelcast --image=hazelcast/hazelcast --env="DNS_DOMAIN=cluster" -
-env="POD_NAMESPACE=default"

# Start a hazelcast pod and set labels "app=hazelcast" and "env=prod" in
the container
ac run hazelcast --image=hazelcast/hazelcast --labels="app=hazelcast,env=
prod"

# Dry run; print the corresponding API objects without creating them
ac run nginx --image=nginx --dry-run=client

# Start a nginx pod, but overload the spec with a partial set of values p
arsed from JSON
ac run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": {
... } }'

# Start a busybox pod and keep it in the foreground, don't restart it if
it exits
ac run -i -t busybox --image=busybox --restart=Never

# Start the nginx pod using the default command, but use custom arguments
(arg1 .. argN) for that command
ac run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# Start the nginx pod using a different command and custom arguments
ac run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>
```

ac scale

Set a new size for a deployment, replica set, or replication controller

Example usage

```
# Scale a replica set named 'foo' to 3
ac scale --replicas=3 rs/foo

# Scale a resource identified by type and name specified in "foo.yaml" to
3
ac scale --replicas=3 -f foo.yaml

# If the deployment named mysql's current size is 2, scale mysql to 3
ac scale --current-replicas=2 --replicas=3 deployment/mysql

# Scale multiple replication controllers
ac scale --replicas=5 rc/example1 rc/example2 rc/example3

# Scale stateful set named 'web' to 3
ac scale --replicas=3 statefulset/web
```

ac set

Set specific features on objects

ac set env

Update environment variables on a pod template

Example usage

```
# Update deployment 'registry' with a new environment variable
ac set env deployment/registry STORAGE_DIR=/local

# List the environment variables defined on a deployments 'sample-build'
ac set env deployment/sample-build --list

# List the environment variables defined on all pods
ac set env pods --all --list

# Output modified deployment in YAML, and does not alter the object on th
e server
ac set env deployment/sample-build STORAGE_DIR=/data -o yaml

# Update all containers in all replication controllers in the project to
have ENV=prod
ac set env rc --all ENV=prod

# Import environment from a secret
ac set env --from=secret/mysecret deployment/myapp

# Import environment from a config map with a prefix
ac set env --from=configmap/myconfigmap --prefix=MYSQL_ deployment/myapp

# Import specific keys from a config map
ac set env --keys=my-example-key --from=configmap/myconfigmap deployment/
myapp

# Remove the environment variable ENV from container 'c1' in all deployme
nt configs
ac set env deployments --all --containers="c1" ENV-

# Remove the environment variable ENV from a deployment definition on dis
k and
# update the deployment config on the server
ac set env -f deploy.json ENV-

# Set some of the local shell environment into a deployment config on the
server
env | grep RAILS_ | ac set env -e - deployment/registry
```

ac set image

Update the image of a pod template

Example usage

```
# Set a deployment's nginx container image to 'nginx:1.9.1', and its busy box container image to 'busybox'
ac set image deployment/nginx busybox=busybox nginx=nginx:1.9.1

# Update all deployments' and rc's nginx container's image to 'nginx:1.9.1'
ac set image deployments,rc nginx=nginx:1.9.1 --all

# Update image of all containers of daemonset abc to 'nginx:1.9.1'
ac set image daemonset abc *=nginx:1.9.1

# Print result (in yaml format) of updating nginx container image from local file, without hitting the server
ac set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml
```

ac set resources

Update resource requests/limits on objects with pod templates

Example usage

```
# Set a deployments nginx container cpu limits to "200m" and memory to "5
12Mi"
ac set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# Set the resource request and limits for all containers in nginx
ac set resources deployment nginx --limits=cpu=200m,memory=512Mi --reques
ts=cpu=100m,memory=256Mi

# Remove the resource requests for resources on containers in nginx
ac set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=
0,memory=0

# Print the result (in yaml format) of updating nginx container limits fr
om a local, without hitting the server
ac set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --lo
cal -o yaml
```

ac set selector

Set the selector on a resource

Example usage

```
# Set the labels and selector before creating a deployment/service pair
ac create service clusterip my-svc --clusterip="None" -o yaml --dry-run=c
lient | 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

Update the service account of a resource

Example usage

```
# Set deployment nginx-deployment's service account to serviceaccount1
ac set serviceaccount deployment nginx-deployment serviceaccount1

# Print the result (in YAML format) of updated nginx deployment with the
service account from local file, without hitting the API server
ac set sa -f nginx-deployment.yaml serviceaccount1 --local --dry-run=clie
nt -o yaml
```

ac set subject

Update the user, group, or service account in a role binding or cluster role binding

Example usage

```
# Update a cluster role binding for serviceaccount1
ac set subject clusterrolebinding admin --serviceaccount=namespace:servic
eaccount1

# Update a role binding for user1, user2, and group1
ac set subject rolebinding admin --user=user1 --user=user2 --group=group1

# Print the result (in YAML format) of updating rolebinding subjects from
a local, without hitting the server
ac create rolebinding admin --role=admin --user=admin -o yaml --dry-run=c
lient | ac set subject --local -f - --user=foo -o yaml
```

ac version

Print the client and server version information

Example usage

```
# Print client and server version information
ac version

# Print client version only
ac version --client

# Print version in JSON format
ac version -o json
```

ac wait

Experimental: Wait for a specific condition on one or many resources

Example usage

```
# Wait for the pod "busybox1" to contain the status condition of type "Ready"
ac wait --for=condition=Ready pod/busybox1

# The default value of status condition is true; you can wait for other targets after an equal delimiter (compared after Unicode simple case folding, which is a more general form of case-insensitivity)
ac wait --for=condition=Ready=false pod/busybox1

# Wait for the pod "busybox1" to contain the status phase to be "Running"
ac wait --for=jsonpath='{.status.phase}'=Running pod/busybox1

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

# Wait for the service "loadbalancer" to have ingress
ac wait --for=jsonpath='{.status.loadBalancer.ingress}' service/loadbalancer

# Wait for the secret "busybox1" to be created, with a timeout of 30s
ac create secret generic busybox1
ac wait --for=create secret/busybox1 --timeout=30s

# Wait for the pod "busybox1" to be deleted, with a timeout of 60s, after having issued the "delete" command
ac delete pod/busybox1
ac wait --for=delete pod/busybox1 --timeout=60s
```

ac whoami

Show information about the current session

Example usage

```
# Display the currently authenticated user
ac whoami

# Print the current user context name
ac whoami -c

# Print the current server's REST API URL
ac whoami --show-server

# Print the token the current session is using
ac whoami -t
# Caution: the token printed by ac whoami -t is sensitive; do not expose
it in logs, shell history, screenshots, or shared output.
```

AC CLI Administrator Command Reference

This reference provides descriptions and example commands for AC CLI administrator commands. You must have cluster-admin or equivalent permissions to use these commands.

For developer commands, see the AC CLI developer command reference.

Run `ac adm -h` to list all administrator commands or run `ac <command> --help` to get additional details for a specific command.

TOC

ac adm
Example usage
ac adm certificate
ac adm certificate approve
Example usage
ac adm certificate deny
Example usage
ac adm cordon
Example usage
ac adm drain
Example usage
ac adm new-project
Example usage
ac adm new-project-namespace

Example usage
ac adm policy
Example usage
ac adm policy add-cluster-role-to-user
Example usage
ac adm policy add-namespace-role-to-user
Example usage
ac adm policy add-project-role-to-user
Example usage
ac adm policy add-role-to-user
Example usage
ac adm release
Example usage
ac adm release import-manifest
Example usage
ac adm taint
Example usage
ac adm uncordon
Example usage
ac adm upgrade
Example usage
ac adm upgrade status
Example usage

ac adm

ACP administrative tools for cluster management

Example usage

```
# Drain a node for maintenance
ac adm drain NODE_NAME

# Cordon a node (mark as unschedulable)
ac adm cordon NODE_NAME

# new-project to create project with cluster
ac adm new-project PROJECT_NAME --cluster CLUSTER_NAME

# Uncordon a node (mark as schedulable)
ac adm uncordon NODE_NAME
```

ac adm certificate

Modify certificate resources

ac adm certificate approve

Approve a certificate signing request

Example usage

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

ac adm certificate deny

Deny a certificate signing request

Example usage

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

ac adm cordon

Mark node as unschedulable

Example usage

```
# Mark node "foo" as unschedulable
ac adm cordon foo
```

ac adm drain

Drain node in preparation for maintenance

Example usage

```
# Drain node "foo", even if there are pods not managed by a replication c
ontroller, replica set, job, daemon set, or stateful set on it
ac adm drain foo --force

# As above, but abort if there are pods not managed by a replication cont
roller, replica set, job, daemon set, or stateful set, and use a grace pe
riod of 15 minutes
ac adm drain foo --grace-period=900
```

ac adm new-project

Create a new project

Example usage

```
# Create a project with specific clusters
ac adm new-project my-project --cluster cluster1

# Create a project with multiple clusters
ac adm new-project my-project --cluster cluster1,cluster2
```

ac adm new-project-namespace

Create a new namespace in project

Example usage

```
# Create a namespace in project with specific clusters
ac adm new-project-namespace  my-namespace --project my-project --cluster
cluster1
```

ac adm policy

Manage RBAC policy with project or namespace

Example usage

```
# Assign a user to the admin role in a project
ac adm policy add-project-role-to-user project-admin-system alice --proje
ct my-project

# Assign a user to the namespace role in a cluster namespace in project
ac adm policy add-namespace-role-to-user namespace-developer-system alice
--namespace my-namespace --project my-project --cluster business-1

# add kubernetes cluster role  view  to user alice
ac adm policy add-cluster-role-to-user view alice

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

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

Assign a kubernetes cluster role to a user in current context cluster

Example usage

```
# add kubernetes cluster role  view  to user alice
ac adm policy add-cluster-role-to-user view alice
```

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

Assign a platform role to a user in a special cluster namespace in project

Example usage

```
# Assign the namespace-developer-system role to user alice in project my-project
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

Assign a platform role to a user in a project

Example usage

```
# Assign the project-admin-system role to user alice in project my-project
ac adm policy add-project-role-to-user project-admin-system alice --project my-project
```

ac adm policy add-role-to-user

Assign a kubernetes role to a user in current context cluster

Example usage

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

ac adm release

Manage release metadata and related administrative workflows

Example usage

```
# Import a ProductManifest for a release version
ac adm release import-manifest --version 4.20.0
```

ac adm release import-manifest

Import release metadata as a ProductManifest

Example usage

```
# Import release metadata for version 4.20.0
ac adm release import-manifest --version 4.20.0

# Import metadata and wait for the ProductManifest to become Ready
ac adm release import-manifest --version 4.20.0 --wait

# Override the wait timeout
ac adm release import-manifest --version 4.20.0 --wait --timeout=10m
```

ac adm taint

Update the taints on one or more nodes

Example usage

```
# Update node 'foo' with a taint with key 'dedicated' and value 'special-user' and effect 'NoSchedule'
# If a taint with that key and effect already exists, its value is replaced as specified
ac adm taint nodes foo dedicated=special-user:NoSchedule

# Remove from node 'foo' the taint with key 'dedicated' and effect 'NoSchedule' if one exists
ac adm taint nodes foo dedicated:NoSchedule-

# Remove from node 'foo' all the taints with key 'dedicated'
ac adm taint nodes foo dedicated-

# Add a taint with key 'dedicated' on nodes having label myLabel=X
ac adm taint node -l myLabel=X  dedicated=foo:PreferNoSchedule

# Add to node 'foo' a taint with key 'bar' and no value
ac adm taint nodes foo bar:NoSchedule
```

ac adm uncordon

Mark node as schedulable

Example usage

```
# Mark node "foo" as schedulable
ac adm uncordon foo
```

ac adm upgrade

Review or request a cluster upgrade

Example usage

```
# View the update status and available cluster updates
ac adm upgrade

# View summary for a specific cluster
ac adm upgrade --cluster=workload-a

# Update to the latest version
ac adm upgrade --to-latest

# Update to a specific version from available updates
ac adm upgrade --cluster=workload-a --to=4.15.0

# Allow an explicit version outside available updates
ac adm upgrade --to=4.15.0 --allow-explicit-upgrade
```

ac adm upgrade status

Review preflight and stage details for the target cluster upgrade

Example usage

```
# Review the status of the Cluster Version Operator for default cluster
ac adm upgrade status

# Review the status of a specific cluster
ac adm upgrade status --cluster=workload-a

# Review the full controller-reported details
ac adm upgrade status --verbose
```

Upgrading Clusters

AC CLI provides administrator commands for preparing upgrade metadata, reviewing cluster upgrade status, and requesting cluster upgrades.

Use these commands when you need to:

- Create a `ProductManifest` for a target version
- Check the current cluster version and available updates
- Request an upgrade to the latest version
- Request an upgrade to a specific version
- Review preflight results and upgrade execution progress

TOC

Before You Start
Prerequisites
Creating a ProductManifest
Viewing Upgrade Status and Available Updates
Updating to the Latest Version
Updating to a Specific Version
Reviewing Detailed Upgrade Status
Preflight Results
Upgrade Stages
Common Signals and Troubleshooting

Example Workflow

Before You Start

In ACP, `availableUpdates` is not a static list that you maintain manually. The upgrade controller must first see a `ProductManifest` for the target version before it can publish available upgrade targets for the cluster.

If you do not create the `ProductManifest` first, common symptoms include:

- `ac adm upgrade` shows no `availableUpdates`
- `ac adm upgrade --to-latest` fails immediately
- You can only use `--allow-explicit-upgrade` to request a version manually

The recommended flow is:

1. Decide which version you want to publish or upgrade to.
2. Create a `ProductManifest` for that version.
3. Wait for the upgrade metadata to be processed.
4. Run `ac adm upgrade` and confirm that `availableUpdates` is populated.
5. Request the upgrade.

Prerequisites

Before you run upgrade-related commands, make sure that:

- You are logged in to an ACP platform.
- You have cluster administrator or equivalent permissions.
- You know the target cluster name. If omitted, `ac adm upgrade` uses `global` by default.
- The target environment already has the `ProductManifest` CRD and the upgrade controller installed.

You can confirm the current context first:

```
ac config current-context
```

The current context still decides which credentials and endpoint AC uses for the command.

- For `ac adm upgrade` and `ac adm upgrade status`, the target cluster is still selected explicitly with `--cluster`, which defaults to `global`.
- For `ac adm release import-manifest`, AC first inspects the current context REST URL. If it points to an ACP workload path, AC rewrites the request to the matching global path automatically. If the current context is not an ACP workload/global URL, AC uses the current context as-is and does not require a kubeconfig session extension.

If needed, switch to another ACP session first:

```
# Switch to another ACP session
ac config use-session production
```

Creating a ProductManifest

Use the following command to create a `ProductManifest` for the target version:

```
ac adm release import-manifest --version 4.20.0
```

This command creates the minimum upgrade metadata object required by the controller:

- `metadata.name` uses the version name with a leading `v`, for example `v4.20.0`
- `spec.version` uses the version you passed in, for example `4.20.0`

If you want the command to wait until the object is Ready, add `--wait`:

```
ac adm release import-manifest --version 4.20.0 --wait
```

You can also override the wait timeout:

```
ac adm release import-manifest --version 4.20.0 --wait --timeout=10m
```

Command behavior:

- `--version` is required.
- If the `ProductManifest` does not exist, AC creates it.
- If the `ProductManifest` already exists with the same version, the command succeeds without changing it.
- If the `ProductManifest` already exists with a different version, the command fails and does not overwrite the existing object.
- By default, the command does not wait for Ready. It waits only when you explicitly pass `--wait`.

Viewing Upgrade Status and Available Updates

After the `ProductManifest` is created, use the following command to review the upgrade summary for the default `global` cluster:

```
ac adm upgrade
```

This command typically shows:

- The current cluster version
- The desired version, if an upgrade has already been requested
- The current list of available updates
- The overall upgrade conditions

Use it when you want quick answers to questions like:

- Which version is the cluster currently running?
- Has a target version already been requested?
- Are new upgrade targets available now?

- Is the cluster currently `Ready`, `Reconciling`, or `Degraded`?

To query a specific cluster, add `--cluster`:

```
ac adm upgrade --cluster=workload-a
```

If you just created a `ProductManifest` and still do not see `availableUpdates`, the controller may still be processing the metadata. Wait a moment and run `ac adm upgrade` again.

Updating to the Latest Version

When `availableUpdates` is present, request an upgrade to the latest version with:

```
ac adm upgrade --to-latest
```

AC selects the highest version from `availableUpdates` and submits the upgrade request.

`--to-latest` is a boolean flag with a default value of `false`, which means:

- If you do not specify it, AC behaves as if `--to-latest=false` was used.
- If you specify `--to-latest` by itself, AC treats it as `true`.
- You can also write `--to-latest=true` or `--to-latest=false` explicitly.

If no `availableUpdates` exist, the command fails and does not submit a new target version.

After you request the upgrade, review the summary again:

```
ac adm upgrade
```

Updating to a Specific Version

If you want to upgrade to a specific version, run:

```
ac adm upgrade --to=<version>
```

Example:

```
ac adm upgrade --to=4.20.0
```

Typical cases include:

- You do not want the newest available version
- You are following an approved release target from your release process
- You want to retry a known target version

By default, the requested version must already appear in `availableUpdates`. If it does not, the command fails.

If you intentionally need to request a version that is not listed in `availableUpdates`, add:

```
ac adm upgrade --to=<version> --allow-explicit-upgrade
```

`--allow-explicit-upgrade` defaults to `false`:

- If you do not specify it, AC behaves as if `--allow-explicit-upgrade=false` was used.
- If you specify `--allow-explicit-upgrade` by itself, AC treats it as `true`.
- You can also write `--allow-explicit-upgrade=true` or `--allow-explicit-upgrade=false` explicitly.

Example:

```
ac adm upgrade --to=4.20.0 --allow-explicit-upgrade=true
```

This flag only changes client-side validation. The upgrade controller and its preflight checks still decide whether the requested target can proceed.

Reviewing Detailed Upgrade Status

If you need deeper diagnostics, run:

```
ac adm upgrade status
```

To review a specific cluster:

```
ac adm upgrade status --cluster=workload-a
```

Compared with `ac adm upgrade`, this command expands:

- A summary of the current version, desired version, and overall conditions
- Preflight results for the current upgrade target
- Upgrade stages and operation progress

Preflight Results

Preflight describes whether the upgrade can move into execution. Each check can typically include:

- Check name
- Reentry policy
- State
- Reason
- Message

Interpret the states as follows:

- `Passed`: The check passed.
- `Retry`: The check cannot produce a final result yet. Wait and check again.
- `Failed`: A blocking condition exists and must be handled first.

If no preflight data is available yet, treat it as "no result yet", not as "all checks passed".

If `AdminAckRequired` is `Failed`, an administrator acknowledgement gate applies to the current upgrade target. Review the gate, complete the required manual checks, write the acknowledgement, and then run `ac adm upgrade status --cluster=<cluster>` again to confirm that the check passes.

Upgrade Stages

After the upgrade enters execution, the status output shows stage and operation progress.

Stage output can include:

- Stage name
- Priority
- Phase

Operation output can include:

- Operation name
- Action
- Current version
- Target version
- Phase

Interpret stage phases as follows:

- `Pending`: The stage has not started yet.
- `Running`: The stage is currently in progress.
- `Finished`: The stage has completed.

If no stage data is available yet, the upgrade likely has not entered the execution phase.

Common Signals and Troubleshooting

Use the following guidelines when you read upgrade status:

- `Ready` usually means the cluster has reached the desired state.
- `Reconciling` usually means the cluster is still applying the current upgrade request.
- `Degraded` usually means the upgrade is blocked or has encountered an error.

When `ac adm upgrade` does not show any `availableUpdates`, check these items first:

1. Was the `ProductManifest` for the target version created?
2. If you used `--wait`, did the `ProductManifest` reach `Ready=True`?
3. Has the controller had enough time to process the new metadata?

When `ac adm upgrade` shows a desired target but the upgrade is not moving:

1. Run `ac adm upgrade status`.
2. Check whether any preflight item is in `Retry` or `Failed`.
3. Review whether upgrade stages have started.

When `ac adm upgrade status --cluster=<cluster>` shows `AdminAckRequired` in `Failed` state:

1. Inspect the release-provided gates from the global cluster:

   ```
   kubectl -n cpaas-system get configmap admin-gates -o yaml
   ```

2. If the target release is ACP 4.4 or later, upgrades the cluster to Kubernetes 1.35 or later, and `admin-gates` reports a node-readiness acknowledgement key, complete the Kubernetes 1.35 or later node readiness checks on every production node in the target cluster. The first gate introduced for this requirement is `ack-4.4-kubernetes-1.35-kernel-update`; treat it as an example because a later target release can provide a different key.
3. After completing the gate requirements, copy the applicable key from `admin-gates` and write it to the global-side `admin-acks` ConfigMap:

   ```
   ACK_KEY='<key-from-admin-gates>'
   kubectl -n cpaas-system patch configmap admin-acks --type merge \
     -p "{\"data\":{\"${ACK_KEY}\":\"true\"}}"
   ```

4. Confirm that the preflight check passes:

```
ac adm upgrade status --cluster=<cluster>
```

When the upgrade is already in progress:

1. Run `ac adm upgrade status`.
2. Review the current stage and operation phases.
3. Compare the current version with the target version.

Example Workflow

```
# 1. Create a ProductManifest for the target version
ac adm release import-manifest --version 4.20.0 --wait

# 2. Review the upgrade summary for the default global cluster
ac adm upgrade

# 3. Review the summary for a specific cluster
ac adm upgrade --cluster=workload-a

# 4. Request an upgrade to the latest available version
ac adm upgrade --to-latest

# 5. Review detailed status
ac adm upgrade status --cluster=workload-a

# 6. Request an upgrade to a specific version
ac adm upgrade --cluster=workload-a --to=4.20.0

# 7. Request a version outside availableUpdates when you explicitly inten
d to do so
ac adm upgrade --cluster=workload-a --to=4.20.0 --allow-explicit-upgrade
```