:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#plugins] = Configure plugins for the CDK Toolkit :info\_titleabbrev: Configure plugins :keywords: CDK Command Line Interface, CDK CLI, {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), credentials, plugins, CDK Toolkit, CDK Toolkit Library

== [abstract]

## The {aws} CDK Toolkit provides a plugin system that enables developers to extend its functionality.

// Content start

The {aws} CDK Toolkit supports plugins that add new capabilities to your CDK workflows. Plugins are primarily designed for use with the CDK Command Line Interface (CDK CLI), though they can also be used with the CDK Toolkit Library programmatically. They provide a standardized way to extend functionality, such as alternative credential sources.

= [NOTE]

# While the plugin system works with both the CDK CLI and CDK Toolkit Library, it is primarily intended for CLI usage. When using the CDK Toolkit Library programmatically, there are often simpler, more direct ways to accomplish the same tasks without plugins, particularly for credential management.

Currently, the CDK Toolkit supports one plugin capability:

* *Custom {aws} credential providers* - Create alternative methods to obtain {aws} credentials beyond the built-in mechanisms.

[#plugins-create] == How to create plugins

To create a CDK Toolkit plugin, you first create a Node.js module, authored in TypeScript or JavaScript, that can be loaded by the CDK Toolkit. This module exports an object with a specific structure that the CDK Toolkit can recognize. At minimum, the exported object must include a version identifier and an initialization function that receives an IPluginHost instance, which the plugin uses to register its capabilities.

==== [role=“tablist”] TypeScript:: + *CommonJS*::: + [source,typescript,subs=“verbatim,attributes”] — // Example plugin structure import type { IPluginHost, Plugin } from ‘@aws-cdk/cli-plugin-contract’;

const plugin: Plugin = { // Version of the plugin infrastructure (currently always ‘1’) version: ‘1’,

// Initialization function called when the plugin is loaded init(host: IPluginHost): void { // Register your plugin functionality with the host // For example, register a custom credential provider } };

== export = plugin;

* *ESM*:::

## [source,typescript,subs=“verbatim,attributes”]

* // Example plugin structure import type { IPluginHost, Plugin } from ‘@aws-cdk/cli-plugin-contract’;

const plugin: Plugin = { // Version of the plugin infrastructure (currently always ‘1’) version: ‘1’,

// Initialization function called when the plugin is loaded init(host: IPluginHost): void { // Register your plugin functionality with the host // For example, register a custom credential provider } };

== export { plugin as ‘module.exports’ };

JavaScript:: + *CommonJS*::: + [source,javascript,subs=“verbatim,attributes”] — const plugin = { // Version of the plugin infrastructure (currently always ‘1’) version: ‘1’,

// Initialization function called when the plugin is loaded init(host) { // Register your plugin functionality with the host // For example, register a custom credential provider } };

== module.exports = plugin;

* *ESM*:::

## [source,javascript,subs=“verbatim,attributes”]

* const plugin = { // Version of the plugin infrastructure (currently always ‘1’) version: ‘1’,
* // Initialization function called when the plugin is loaded init(host) { // Register your plugin functionality with the host // For example, register a custom credential provider } };

== export { plugin as ‘module.exports’ };

====

= [NOTE]

CDK Toolkit plugins are loaded as CommonJS modules. You can build your plugin as an ECMAScript module (ESM), but have to link:https://nodejs.org/api/modules.html#loading-ecmascript-modules-using-require[adhere to some constraints]:

–

* The module cannot contain top-level await, or import any modules that contain top-level await.
* {blank}
* == The module must ensure compatibility by exporting the plugin object under the module.exports export name: export { plugin as 'module.exports' }.
* ====

[#plugins-load-cli] == How to load plugins with the CDK CLI

You have two options to specify plugins:

*Option 1: Use the --plugin command line option*:: + [source,bash,subs=“verbatim,attributes”] —

= Load a single plugin

$ cdk list –plugin=my-custom-plugin

= Load multiple plugins

## $ cdk deploy –plugin=custom-plugin-1 –plugin=custom-plugin-2

* The value of the --plugin argument should be a JavaScript file that, when imported via the Node.js require() function, returns an object implementing the Plugin interface.

*Option 2: Add entries to configuration files*:: + You can add plugin specifications to the project-specific cdk.json file or the ~/.cdk.json global configuration file: + [source,json,subs=“verbatim,attributes”] — { “plugin”: [ “custom-plugin-1”, “custom-plugin-2” ] } — + With either approach, the CDK CLI will load the specified plugins before running commands.

[#plugins-load-library] === Load plugins in code with the CDK Toolkit Library

When working with the CDK Toolkit Library programmatically, you can load plugins directly in your code using the PluginHost instance from the @aws-cdk/toolkit-lib package. The PluginHost provides a load() method for loading plugins by module name or path.

==== [role=“tablist”] TypeScript:: + [source,typescript,subs=“verbatim,attributes”] — import { Toolkit } from ‘@aws-cdk/toolkit-lib’;

// Create a Toolkit instance const toolkit = new Toolkit();

// Load a plugin by module name or path // The module must export an object matching the Plugin interface await toolkit.pluginHost.load(‘my-custom-plugin’);

// You can load multiple plugins if needed await toolkit.pluginHost.load(‘./path/to/another-plugin’);

// Now proceed with other CDK Toolkit Library operations // The plugin’s functionality will be available to the toolkit —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { Toolkit } = require(‘@aws-cdk/toolkit-lib’);

// Create a Toolkit instance const toolkit = new Toolkit();

// Load a plugin by module name or path // The module must export an object matching the Plugin interface await toolkit.pluginHost.load(‘my-custom-plugin’);

// You can load multiple plugins if needed await toolkit.pluginHost.load(‘./path/to/another-plugin’);

// Now proceed with other CDK Toolkit Library operations // The plugin’s functionality will be available to the toolkit — ====

The load() method takes a single parameter, moduleSpec, which is the name or path of the plugin module to load. This can be either:

* A Node.js module name installed in the node\_modules directory.
* A relative or absolute file path to a JavaScript or TypeScript module.

[#plugins-credentials] == Implementing credential provider plugins

The current primary use case for plugins is to create custom {aws} credential providers. Like the {aws} CLI, the CDK CLI needs {aws} credentials for authentication and authorization. However, there are several scenarios where the standard credential resolution might fail:

* The initial set of credentials cannot be obtained.
* The account to which the initial credentials belong cannot be obtained.
* The account associated with the credentials is different from the account on which the CLI is trying to operate.

To address these scenarios, the CDK Toolkit supports credential provider plugins. These plugins implement the CredentialProviderSource interface from the @aws-cdk/cli-plugin-contract package and are registered to the Toolkit using the registerCredentialProviderSource method. This enables the CDK Toolkit to obtain {aws} credentials from non-standard sources like specialized authentication systems or custom credential stores.

To implement a custom credential provider, create a class that implements the required interface:

==== [role=“tablist”] TypeScript:: + *CommonJS*::: + [source,typescript,subs=“verbatim,attributes”] — import type { CredentialProviderSource, ForReading, ForWriting, IPluginHost, Plugin, PluginProviderResult, SDKv3CompatibleCredentials } from ‘@aws-cdk/cli-plugin-contract’;

class CustomCredentialProviderSource implements CredentialProviderSource { // Friendly name for the provider, used in error messages public readonly name: string = ‘custom-credential-provider’;

// Check if this provider is available on the current system public async isAvailable(): Promise++++++{ // Return false if the plugin cannot be used // For example, if it depends on files not present on the host return true; }++++++

// Check if this provider can provide credentials for a specific account public async canProvideCredentials(accountId: string): Promise++++++{ // Return false if the plugin cannot provide credentials for this account // For example, if the account is not managed by this credential system return true; // You can use patterns to filter specific accounts // return accountId.startsWith(‘123456’); }++++++

// Get credentials for the specified account and access mode // Returns PluginProviderResult which can be one of: // - SDKv2CompatibleCredentials (AWS SDK v2 entered maintenance on Sept 8, 2024 and will reach end-of-life on Sept 8, 2025) // - SDKv3CompatibleCredentialProvider // - SDKv3CompatibleCredentials public async getProvider(accountId: string, mode: ForReading | ForWriting): Promise++++++{ // The access mode can be used to provide different credential sets const readOnly = mode === 0 satisfies ForReading;++++++

…. // Create appropriate credentials based on your authentication mechanism const credentials: SDKv3CompatibleCredentials = { accessKeyId: ‘AKIAIOSFODNN7EXAMPLE’, secretAccessKey: ‘wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY’, // Add sessionToken if using temporary credentials // sessionToken: ‘AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk’, // expireTime: new Date(Date.now() + 3600 \* 1000), // 1 hour from now };

return credentials; } } ….

const plugin: Plugin = { version: ‘1’, init(host: IPluginHost): void { // Register the credential provider to the PluginHost. host.registerCredentialProviderSource(new CustomCredentialProviderSource()); } };

== export = plugin;

* *ESM*:::

## [source,typescript,subs=“verbatim,attributes”]

* import type { CredentialProviderSource, ForReading, ForWriting, IPluginHost, Plugin, PluginProviderResult, SDKv3CompatibleCredentials } from ‘@aws-cdk/cli-plugin-contract’;

class CustomCredentialProviderSource implements CredentialProviderSource { // Friendly name for the provider, used in error messages public readonly name: string = ‘custom-credential-provider’;

// Check if this provider is available on the current system public async isAvailable(): Promise++++++{ // Return false if the plugin cannot be used // For example, if it depends on files not present on the host return true; }++++++

// Check if this provider can provide credentials for a specific account public async canProvideCredentials(accountId: string): Promise++++++{ // Return false if the plugin cannot provide credentials for this account // For example, if the account is not managed by this credential system return true; // You can use patterns to filter specific accounts // return accountId.startsWith(‘123456’); }++++++

// Get credentials for the specified account and access mode // Returns PluginProviderResult which can be one of: // - SDKv2CompatibleCredentials (AWS SDK v2 entered maintenance on Sept 8, 2024 and will reach end-of-life on Sept 8, 2025) // - SDKv3CompatibleCredentialProvider // - SDKv3CompatibleCredentials public async getProvider(accountId: string, mode: ForReading | ForWriting): Promise++++++{ // The access mode can be used to provide different credential sets const readOnly = mode === 0 satisfies ForReading;++++++

…. // Create appropriate credentials based on your authentication mechanism const credentials: SDKv3CompatibleCredentials = { accessKeyId: ‘AKIAIOSFODNN7EXAMPLE’, secretAccessKey: ‘wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY’, // Add sessionToken if using temporary credentials // sessionToken: ‘AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk’, // expireTime: new Date(Date.now() + 3600 \* 1000), // 1 hour from now };

return credentials; } } ….

const plugin: Plugin = { version: ‘1’, init(host: IPluginHost): void { // Register the credential provider to the PluginHost. host.registerCredentialProviderSource(new CustomCredentialProviderSource()); } };

== export { plugin as ‘module.exports’ };

JavaScript:: + *CommonJS*::: + [source,javascript,subs=“verbatim,attributes”] — // Implement the CredentialProviderSource interface class CustomCredentialProviderSource { constructor() { // Friendly name for the provider, used in error messages this.name = ‘custom-credential-provider’; }

// Check if this provider is available on the current system async isAvailable() { // Return false if the plugin cannot be used // For example, if it depends on files not present on the host return true; }

// Check if this provider can provide credentials for a specific account async canProvideCredentials(accountId) { // Return false if the plugin cannot provide credentials for this account // For example, if the account is not managed by this credential system return true; // You can use patterns to filter specific accounts // return accountId.startsWith(‘123456’); }

// Get credentials for the specified account and access mode // Returns PluginProviderResult which can be one of: // - SDKv2CompatibleCredentials (AWS SDK v2 entered maintenance on Sept 8, 2024 and will reach end-of-life on Sept 8, 2025) // - SDKv3CompatibleCredentialProvider // - SDKv3CompatibleCredentials async getProvider(accountId, mode) { // The access mode can be used to provide different credential sets const readOnly = mode === 0; // 0 indicates ForReading; 1 indicates ForWriting

…. // Create appropriate credentials based on your authentication mechanism const credentials = { accessKeyId: ‘ASIAIOSFODNN7EXAMPLE’, secretAccessKey: ‘wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY’, // Add sessionToken if using temporary credentials // sessionToken: ‘AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk’, // expireTime: new Date(Date.now() + 3600 \* 1000), // 1 hour from now };

return credentials; } } ….

const plugin = { version: ‘1’, init(host) { // Register the credential provider to the PluginHost. host.registerCredentialProviderSource(new CustomCredentialProviderSource()); } };

== module.exports = plugin;

* *ESM*:::

## [source,javascript,subs=“verbatim,attributes”]

* // Implement the CredentialProviderSource interface class CustomCredentialProviderSource { constructor() { // Friendly name for the provider, used in error messages this.name = ‘custom-credential-provider’; }

// Check if this provider is available on the current system async isAvailable() { // Return false if the plugin cannot be used // For example, if it depends on files not present on the host return true; }

// Check if this provider can provide credentials for a specific account async canProvideCredentials(accountId) { // Return false if the plugin cannot provide credentials for this account // For example, if the account is not managed by this credential system return true; // You can use patterns to filter specific accounts // return accountId.startsWith(‘123456’); }

// Get credentials for the specified account and access mode // Returns PluginProviderResult which can be one of: // - SDKv2CompatibleCredentials (AWS SDK v2 entered maintenance on Sept 8, 2024 and will reach end-of-life on Sept 8, 2025) // - SDKv3CompatibleCredentialProvider // - SDKv3CompatibleCredentials async getProvider(accountId, mode) { // The access mode can be used to provide different credential sets const readOnly = mode === 0; // 0 indicates ForReading; 1 indicates ForWriting

…. // Create appropriate credentials based on your authentication mechanism const credentials = { accessKeyId: ‘ASIAIOSFODNN7EXAMPLE’, secretAccessKey: ‘wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY’, // Add sessionToken if using temporary credentials // sessionToken: ‘AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk’, // expireTime: new Date(Date.now() + 3600 \* 1000), // 1 hour from now };

return credentials; } } ….

const plugin = { version: ‘1’, init(host) { // Register the credential provider to the PluginHost. host.registerCredentialProviderSource(new CustomCredentialProviderSource()); } };

== export { plugin as ‘module.exports’ };

====

= [IMPORTANT]

# Credentials obtained from providers are cached by the CDK Toolkit. It’s strongly recommended that credential objects returned by your provider are self-refreshing to prevent expiration issues during long-running operations. For more information, see link:https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/Package/-aws-sdk-client-sts/Interface/Credentials/[Credentials] in the *{aws} SDK for JavaScript v3 documentation*.

[#plugins-learn] == Learn more

To learn more about CDK Toolkit plugins, see the link:https://github.com/aws/aws-cdk-cli/tree/main/packages/%40aws-cdk/cli-plugin-contract[{aws} CDK Toolkit Plugin Contract] in the *aws-cdk-cli GitHub repository*.