:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#policy-validation-synthesis] = {aws} CDK policy validation at synthesis time :info\_titleabbrev: Policy validation :keywords: cdk, policy, validation

== [abstract]

## By using the appropriate policy validation plugin, you can make the {aws} CDK application check the generated {aws} CloudFormation template against your policies immediately after synthesis.

// Content start

[#policy-validation] == Policy validation at synthesis time

If you or your organization use any policy validation tool, such as https://docs.aws.amazon.com/cfn-guard/latest/ug/what-is-guard.html[{aws} CloudFormation Guard] or https://www.openpolicyagent.org/[OPA], to define constraints on your {aws} CloudFormation template, you can integrate them with the {aws} CDK at synthesis time. By using the appropriate policy validation plugin, you can make the {aws} CDK application check the generated {aws} CloudFormation template against your policies immediately after synthesis. If there are any violations, the synthesis will fail and a report will be printed to the console.

The validation performed by the {aws} CDK at synthesis time validate controls at one point in the deployment lifecycle, but they can’t affect actions that occur outside synthesis. Examples include actions taken directly in the console or via service APIs. They aren’t resistant to alteration of {aws} CloudFormation templates after synthesis. Some other mechanism to validate the same rule set more authoritatively should be set up independently, like https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/hooks.html[{aws} CloudFormation hooks] or https://docs.aws.amazon.com/config/latest/developerguide/WhatIsConfig.html[{aws} Config]. Nevertheless, the ability of the {aws} CDK to evaluate the rule set during development is still useful as it will improve detection speed and developer productivity.

The goal of {aws} CDK policy validation is to minimize the amount of set up needed during development, and make it as easy as possible.

= [NOTE]

This feature is considered experimental, and both the plugin API and the format of the validation report are subject to change in the future.

====

[#for-application-developers] == For application developers

To use one or more validation plugins in your application, use the policyValidationBeta1 property of Stage:

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

import { CfnGuardValidator } from ‘@cdklabs/cdk-validator-cfnguard’; const app = new App({ policyValidationBeta1: [ new CfnGuardValidator() ], }); // only apply to a particular stage const prodStage = new Stage(app, ‘ProdStage’, { policyValidationBeta1: […], }); —

Immediately after synthesis, all plugins registered this way will be invoked to validate all the templates generated in the scope you defined. In particular, if you register the templates in the App object, all templates will be subject to validation.

= [WARNING]

Other than modifying the cloud assembly, plugins can do anything that your {aws} CDK application can. They can read data from the filesystem, access the network etc. It’s your responsibility as the consumer of a plugin to verify that it’s secure to use.

====

[#cfnguard-plugin] === {aws} CloudFormation Guard plugin

Using the link:https://github.com/cdklabs/cdk-validator-cfnguard[CfnGuardValidator] plugin allows you to use https://github.com/aws-cloudformation/cloudformation-guard[{aws} CloudFormation Guard] to perform policy validations. The CfnGuardValidator plugin comes with a select set of https://docs.aws.amazon.com/controltower/latest/userguide/proactive-controls.html[{aws} Control Tower proactive controls] built in. The current set of rules can be found in the https://github.com/cdklabs/cdk-validator-cfnguard/blob/main/README.md[project documentation]. As mentioned in xref:policy-validation[Policy validation at synthesis time], we recommend that organizations set up a more authoritative method of validation using https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/hooks.html[{aws} CloudFormation hooks].

For link:https://docs.aws.amazon.com/controltower/latest/userguide/what-is-control-tower.html[{aws} Control Tower] customers, these same proactive controls can be deployed across your organization. When you enable {aws} Control Tower proactive controls in your {aws} Control Tower environment, the controls can stop the deployment of non-compliant resources deployed via {aws} CloudFormation. For more information about managed proactive controls and how they work, see the https://docs.aws.amazon.com/controltower/latest/userguide/proactive-controls.html[{aws} Control Tower documentation].

These {aws} CDK bundled controls and managed {aws} Control Tower proactive controls are best used together. In this scenario you can configure this validation plugin with the same proactive controls that are active in your {aws} Control Tower cloud environment. You can then quickly gain confidence that your {aws} CDK application will pass the {aws} Control Tower controls by running cdk synth locally.

[#validation-report] === Validation Report

When you synthesize the {aws} CDK app the validator plugins will be called and the results will be printed. An example report is showing below.

== [source,none,subs=“verbatim,attributes”]

## Validation Report (CfnGuardValidator)

(Summary) ╔═══════════╤════════════════════════╗ ║ Status │ failure ║ ╟───────────┼────────────────────────╢ ║ Plugin │ CfnGuardValidator ║ ╚═══════════╧════════════════════════╝ (Violations) Ensure S3 Buckets are encrypted with a KMS CMK (1 occurrences) Severity: medium Occurrences:

…. - Construct Path: MyStack/MyCustomL3Construct/Bucket - Stack Template Path: ./cdk.out/MyStack.template.json - Creation Stack: └── MyStack (MyStack) │ Library: aws-cdk-lib.Stack │ Library Version: 2.50.0 │ Location: Object. (/home/johndoe/tmp/cdk-tmp-app/src/main.ts:25:20) └── MyCustomL3Construct (MyStack/MyCustomL3Construct) │ Library: N/A - (Local Construct) │ Library Version: N/A │ Location: new MyStack (/home/johndoe/tmp/cdk-tmp-app/src/main.ts:15:20) └── Bucket (MyStack/MyCustomL3Construct/Bucket) │ Library: aws-cdk-lib/aws-s3.Bucket │ Library Version: 2.50.0 │ Location: new MyCustomL3Construct (/home/johndoe/tmp/cdk-tmp-app/src/main.ts:9:20) - Resource Name: amzn-s3-demo-bucket - Locations: > BucketEncryption/ServerSideEncryptionConfiguration/0/ServerSideEncryptionByDefault/SSEAlgorithm Recommendation: Missing value for key SSEAlgorithm - must specify aws:kms How to fix: > Add to construct properties for cdk-app/MyStack/Bucket encryption: BucketEncryption.KMS ….

== Validation failed. See above reports for details

By default, the report will be printed in a human readable format. If you want a report in JSON format, enable it using the @aws-cdk/core:validationReportJson via the CLI or passing it directly to the application:

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

const app = new App({ context: { ‘@aws-cdk/core:validationReportJson’: true }, }); —

Alternatively, you can set this context key-value pair using the cdk.json or cdk.context.json files in your project directory (see xref:context[Context values and the {aws} CDK]).

If you choose the JSON format, the {aws} CDK will print the policy validation report to a file called policy-validation-report.json in the cloud assembly directory. For the default, human-readable format, the report will be printed to the standard output.

[#plugin-authors] == For plugin authors

[#plugins] === Plugins

The {aws} CDK core framework is responsible for registering and invoking plugins and then displaying the formatted validation report. The responsibility of the plugin is to act as the translation layer between the {aws} CDK framework and the policy validation tool. A plugin can be created in any language supported by {aws} CDK. If you are creating a plugin that might be consumed by multiple languages then it’s recommended that you create the plugin in TypeScript so that you can use JSII to publish the plugin in each {aws} CDK language.

[#creating-plugins] === Creating plugins

The communication protocol between the {aws} CDK core module and your policy tool is defined by the IPolicyValidationPluginBeta1 interface. To create a new plugin you must write a class that implements this interface. There are two things you need to implement: the plugin name (by overriding the name property), and the validate() method.

The framework will call validate(), passing an IValidationContextBeta1 object. The location of the templates to be validated is given by templatePaths. The plugin should return an instance of ValidationPluginReportBeta1. This object represents the report that the user wil receive at the end of the synthesis.

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

validate(context: IPolicyValidationContextBeta1): PolicyValidationReportBeta1 { // First read the templates using context.templatePaths… // …then perform the validation, and then compose and return the report. // Using hard-coded values here for better clarity: return { success: false, violations: [{ ruleName: ‘CKV\_AWS\_117’, description: ‘Ensure that {aws} Lambda function is configured inside a VPC’, fix: ‘https://docs.bridgecrew.io/docs/ensure-that-aws-lambda-function-is-configured-inside-a-vpc-1’, violatingResources: [{ resourceName: ‘MyFunction3BAA72D1’, templatePath: ‘/home/johndoe/myapp/cdk.out/MyService.template.json’, locations: ‘Properties/VpcConfig’, }], }], }; } —

Note that plugins aren’t allowed to modify anything in the cloud assembly. Any attempt to do so will result in synthesis failure.

If your plugin depends on an external tool, keep in mind that some developers may not have that tool installed in their workstations yet. To minimize friction, we highly recommend that you provide some installation script along with your plugin package, to automate the whole process. Better yet, run that script as part of the installation of your package. With npm, for example, you can add it to the postinstall link:https://docs.npmjs.com/cli/v9/using-npm/scripts[script] in the package.json file.

[#handling-exemptions] === Handling Exemptions

If your organization has a mechanism for handling exemptions, it can be implemented as part of the validator plugin.

An example scenario to illustrate a possible exemption mechanism:

* An organization has a rule that public Amazon S3 buckets aren’t allowed, *except* for under certain scenarios.
* A developer is creating an Amazon S3 bucket that falls under one of those scenarios and requests an exemption (create a ticket for example).
* Security tooling knows how to read from the internal system that registers exemptions

In this scenario the developer would request an exception in the internal system and then will need some way of “registering” that exception. Adding on to the guard plugin example, you could create a plugin that handles exemptions by filtering out the violations that have a matching exemption in an internal ticketing system.

See the existing plugins for example implementations.

* https://github.com/cdklabs/cdk-validator-cfnguard[@cdklabs/cdk-validator-cfnguard]