:doctype: book

include::attributes.txt[]

// Attributes [.topic] [#blueprints] = Configure constructs with CDK Blueprints :keywords: {aws} CDK, {aws} CloudFormation, IaC, Infrastructure as code, CDK projects, {aws} CDK concepts, blueprints, property injection, construct properties, organizational standards

== [abstract]

## CDK Blueprints allow organizations to standardize L2 construct configurations through property injection, making it easier for developers to follow best practices while maintaining flexibility.

// Content start

= [NOTE]

CDK Blueprints is in preview release and is subject to change.

====

Use {aws} CDK Blueprints to standardize and distribute L2 construct configurations across your organization. With Blueprints, you can ensure that {aws} resources are configured consistently according to your organizational standards and best practices. For example, you can automatically enable encryption for all Amazon S3 buckets, apply specific logging configurations to all {aws} Lambda functions, or enforce standard security rules for all security groups.

Blueprints are powered by *property injection*, a mechanism introduced in {aws} CDK link:https://github.com/aws/aws-cdk/releases/tag/v2.196.0[v2.196.0] that allows you to modify construct properties at instantiation time. A Blueprint is a collection of property injectors, where each property injector specifies optimal configuration for a specific L2 construct. The Blueprint represents the overall best practices for your organization.

Blueprints are not a compliance enforcement mechanism. Developers can still override the defaults if needed. For strict compliance enforcement, consider using {aws} CloudFormation Guard, Service Control Policies, or CDK Aspects in addition to Blueprints.

For detailed implementation information, see the link:https://github.com/aws/aws-cdk-rfcs/blob/main/text/0693-property-injection.md[Property Injection RFC].

[#blueprints-key-pieces] == Key components of Blueprints

Blueprints are collections of property injectors that apply default properties to constructs when they’re instantiated. A property injector is a component that implements the IPropertyInjector interface, which intercepts construct creation and modifies or adds properties before the construct is created.

* *IPropertyInjector* - An +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.IPropertyInjector.html[IPropertyInjector]+ defines a way to inject additional properties that are not specified in the props. It is specific to one L2 construct and operates on that construct’s properties.
* *PropertyInjectors* - +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.PropertyInjectors.html[PropertyInjectors]+ are a collection of injectors attached to the construct tree. Injectors can be attached to any construct, but in practice we expect most of them will be attached to App, Stage or Stack.

[#blueprints-use-cases] == Common use cases for Blueprints

You can use CDK Blueprints to standardize many aspects of your AWS resources. Here are some common use cases:

*Security standards*:: + –

* Ensure all Amazon S3 buckets have server-side encryption enabled.
* Configure all security groups to block public access by default.
* Apply least-privilege AWS Identity and Access Management (IAM) permissions to {aws} Lambda functions.
* {blank}
* == Enforce SSL for all network communications.

*Operational excellence*:: + –

* Configure standardized logging for all {aws} Lambda functions.
* Apply consistent tagging strategies across resources.
* Set up default monitoring and alerting thresholds.
* {blank}
* == Implement standard retention policies for logs and backups.

*Cost optimization*:: + –

* Configure appropriate instance sizes based on environment.
* Apply auto-scaling policies with organizational defaults.
* Set lifecycle rules for Amazon S3 buckets to transition objects to cheaper storage classes.
* {blank}
* == Configure default provisioned throughput for databases.

*Compliance requirements*:: + –

* Implement required encryption settings for regulated data.
* Apply necessary backup policies for data retention requirements.
* Configure default Amazon VPC settings that meet security requirements.
* {blank}
* == Ensure resources have required tags for cost allocation.

*Developer productivity*:: + –

* Provide sensible defaults that reduce the need for boilerplate code.
* Create organization-specific Stack classes with built-in injectors.
* Share best practices across teams through reusable injectors.
* {blank}
* == Simplify onboarding by encoding organizational knowledge in code.

[#blueprints-getting-started] == Getting started with Blueprints

Here’s a simple example of how to create and use a property injector:

First, create a property injector for Amazon S3 buckets:

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

import { IPropertyInjector, InjectionContext } from ‘aws-cdk-lib’; import { Bucket, BucketProps, BlockPublicAccess } from ‘aws-cdk-lib/aws-s3’;

export class SecureBucketDefaults implements IPropertyInjector { public readonly constructUniqueId: string;

constructor() { this.constructUniqueId = Bucket.PROPERTY\_INJECTION\_ID; }

public inject(originalProps: BucketProps, \_context: InjectionContext): BucketProps { return { // Set security defaults blockPublicAccess: BlockPublicAccess.BLOCK\_ALL, enforceSSL: true,

// Include original props to allow overrides …originalProps, }; } } —-

Then, use the injector in your CDK application:

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

import { App, Stack } from ‘aws-cdk-lib’; import { Bucket } from ‘aws-cdk-lib/aws-s3’; import { SecureBucketDefaults } from ‘./secure-bucket-defaults’;

// Attach injectors when creating the App const app = new App({ propertyInjectors: [new SecureBucketDefaults()] });

const stack = new Stack(app, ‘MyStack’);

// This bucket automatically gets the default properties const myBucket = new Bucket(stack, ‘MyBucket’); —

Alternatively, you can use the PropertyInjectors.of() method:

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

import { App, Stack, PropertyInjectors } from ‘aws-cdk-lib’; import { SecureBucketDefaults } from ‘./secure-bucket-defaults’;

const app = new App(); PropertyInjectors.of(app).add(new SecureBucketDefaults());

const stack = new Stack(app, ‘MyStack’); const myBucket = new Bucket(stack, ‘MyBucket’); —

[#blueprints-best-practices] == Best practices

* Place default properties before +...originalProps+ to allow overrides.
* Place forced properties after +...originalProps+ to prevent overrides.
* Use a skip flag when creating resources to prevent infinite recursion. For an example, see link:https://github.com/aws/aws-cdk-rfcs/blob/main/text/0693-property-injection.md#what-happens-when-you-need-to-create-a-accesslogbucket-for-a-bucket[What happens when you need to create a accessLogBucket for a Bucket?] in the *Property Injection RFC*.
* Add logging for debugging.
* Use CDK context to enable/disable injectors for testing.

For more detailed information about property injection, including implementation details, troubleshooting tips, and reference information, see the link:https://github.com/aws/aws-cdk-rfcs/blob/main/text/0693-property-injection.md[Property Injection RFC].