[#tokens-number-pass] === Pass number-encoded tokens

When you pass number-encoded tokens to other constructs, it may make sense to convert them to strings first. For example, if you want to use the value of a number-encoded string as part of a concatenated string, converting it helps with readability.

In the following example,portToken is a number-encoded token that we want to pass to our Lambda function as part of connectionString:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { Stack, Duration, CfnOutput, StackProps } from ‘aws-cdk-lib’; // … import \* as lambda from ‘aws-cdk-lib/aws-lambda’;

export class CdkDemoAppStack extends Stack { constructor(scope: Construct, id: string, props?: StackProps) { super(scope, id, props);

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) const portToken = dbCluster.clusterEndpoint.port;

// …

// Example connection string with the port token as a number const connectionString = jdbc:mysql://mydb.cluster.amazonaws.com:${portToken}/mydatabase;

// Use the connection string as an environment variable in a Lambda function const myFunction = new lambda.Function(this, ‘MyLambdaFunction’, { runtime: lambda.Runtime.NODEJS\_20\_X, handler: ‘index.handler’, code: lambda.Code.fromInline(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello World!'), }; };), environment: { DATABASE\_CONNECTION\_STRING: connectionString, // Using the port token as part of the string }, });

// Output the value of our connection string at synthesis console.log(“connectionString:” + connectionString);

// Output the connection string new CfnOutput(this, ‘ConnectionString’, { value: connectionString, }); } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { Stack, Duration, CfnOutput } = require(‘aws-cdk-lib’); // … const lambda = require(‘aws-cdk-lib/aws-lambda’);

class CdkDemoAppStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) const portToken = dbCluster.clusterEndpoint.port;

// …

// Example connection string with the port token as a number const connectionString = jdbc:mysql://mydb.cluster.amazonaws.com:${portToken}/mydatabase;

// Use the connection string as an environment variable in a Lambda function const myFunction = new lambda.Function(this, ‘MyLambdaFunction’, { runtime: lambda.Runtime.NODEJS\_20\_X, handler: ‘index.handler’, code: lambda.Code.fromInline(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello World!'), }; };), environment: { DATABASE\_CONNECTION\_STRING: connectionString, // Using the port token as part of the string }, });

// Output the value of our connection string at synthesis console.log(“connectionString:” + connectionString);

// Output the connection string new CfnOutput(this, ‘ConnectionString’, { value: connectionString, }); } } ….

== module.exports = { CdkDemoAppStack }

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( Duration, Stack, CfnOutput, ) from aws\_cdk import aws\_lambda as \_lambda

= …

class CdkDemoAppStack(Stack):

…. def **init**(self, scope: Construct, construct\_id: str, \*\*kwargs) -> None: super().\_\_init\_\_(scope, construct\_id, \*\*kwargs)

# Define a new VPC  
# ...  
  
# Define an RDS database cluster  
# ...  
  
# Get the port token (this is a token encoded as a number)  
port\_token = db\_cluster.cluster\_endpoint.port  
  
# ...  
  
# Example connection string with the port token as a number  
connection\_string = f"jdbc:mysql://mydb.cluster.amazonaws.com:{port\_token}/mydatabase"  
  
# Use the connection string as an environment variable in a Lambda function  
my\_function = \_lambda.Function(self, 'MyLambdaFunction',  
 runtime=\_lambda.Runtime.NODEJS\_20\_X,  
 handler='index.handler',  
 code=\_lambda.Code.from\_inline("""  
 exports.handler = async function(event) {  
 return {  
 statusCode: 200,  
 body: JSON.stringify('Hello World!'),  
 };  
 };  
 """),  
 environment={  
 'DATABASE\_CONNECTION\_STRING': connection\_string # Using the port token as part of the string  
 }  
)  
  
# Output the value of our connection string at synthesis  
print(f"connectionString: {connection\_string}")  
  
# Output the connection string  
CfnOutput(self, 'ConnectionString',  
 value=connection\_string  
) ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — // … import software.amazon.awscdk.CfnOutput; import software.amazon.awscdk.services.lambda.Function; import software.amazon.awscdk.services.lambda.Runtime; import software.amazon.awscdk.services.lambda.Code;

import java.util.Map;

public class CdkDemoAppStack extends Stack { public CdkDemoAppStack(final Construct scope, final String id) { this(scope, id, null); }

…. public CdkDemoAppStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// Define a new VPC  
// ...  
  
// Define an RDS database cluster  
// ...  
  
// Get the port token (this is a token encoded as a number)  
Number portToken = dbCluster.getClusterEndpoint().getPort();  
  
// ...  
  
// Example connection string with the port token as a number  
String connectionString = "jdbc:mysql://mydb.cluster.amazonaws.com:" + portToken + "/mydatabase";  
  
// Use the connection string as an environment variable in a Lambda function  
Function myFunction = Function.Builder.create(this, "MyLambdaFunction")  
 .runtime(Runtime.NODEJS\_20\_X)  
 .handler("index.handler")  
 .code(Code.fromInline(  
 "exports.handler = async function(event) {\n" +  
 " return {\n" +  
 " statusCode: 200,\n" +  
 " body: JSON.stringify('Hello World!'),\n" +  
 " };\n" +  
 "};"))  
 .environment(Map.of(  
 "DATABASE\_CONNECTION\_STRING", connectionString // Using the port token as part of the string  
 ))  
 .build();  
  
// Output the value of our connection string at synthesis  
System.out.println("connectionString: " + connectionString);  
  
// Output the connection string  
CfnOutput.Builder.create(this, "ConnectionString")  
 .value(connectionString)  
 .build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — // … using Amazon.CDK.AWS.Lambda; using Amazon.CDK.AWS.RDS; using Amazon.CDK; using Constructs; using System; using System.Collections.Generic;

namespace CdkDemoApp { public class CdkDemoAppStack : Stack { internal CdkDemoAppStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Define a new VPC // …

…. // Define an RDS database cluster var dbCluster = new DatabaseCluster(this, “MyRDSCluster”, new DatabaseClusterProps { // … properties would go here });

// Get the port token (this is a token encoded as a number)  
 var portToken = dbCluster.ClusterEndpoint.Port;  
  
 // ...  
  
 // Example connection string with the port token as a number  
 var connectionString = $"jdbc:mysql://mydb.cluster.amazonaws.com:{portToken}/mydatabase";  
  
 // Use the connection string as an environment variable in a Lambda function  
 var myFunction = new Function(this, "MyLambdaFunction", new FunctionProps  
 {  
 Runtime = Runtime.NODEJS\_20\_X,  
 Handler = "index.handler",  
 Code = Code.FromInline(@"  
 exports.handler = async function(event) {  
 return {  
 statusCode: 200,  
 body: JSON.stringify('Hello World!'),  
 };  
 };  
 "),  
 Environment = new Dictionary<string, string>  
 {  
 { "DATABASE\_CONNECTION\_STRING", connectionString } // Using the port token as part of the string  
 }  
 });  
  
 // Output the value of our connection string at synthesis  
 Console.WriteLine($"connectionString: {connectionString}");  
  
 // Output the connection string  
 new CfnOutput(this, "ConnectionString", new CfnOutputProps  
 {  
 Value = connectionString  
 });  
}

} } ….

’’’

Go:: + [source,go,subs=“verbatim,attributes”] — // … “github.com/aws/aws-cdk-go/awscdk/v2/awslambda” )

type CdkDemoAppStackProps struct { awscdk.StackProps }

func NewCdkDemoAppStack(scope constructs.Construct, id string, props \*CdkDemoAppStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) portToken := dbCluster.ClusterEndpoint().Port()

// …

// Example connection string with the port token as a number connectionString := fmt.Sprintf(“jdbc:mysql://mydb.cluster.amazonaws.com:%s/mydatabase”, portToken)

// Use the connection string as an environment variable in a Lambda function myFunction := awslambda.NewFunction(stack, jsii.String(“MyLambdaFunction”), &awslambda.FunctionProps{ Runtime: awslambda.Runtime\_NODEJS\_20\_X(), Handler: jsii.String(“index.handler”), Code: awslambda.Code\_FromInline(jsii.String(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello World!'), }; };)), Environment: &map[string]\*string{ “DATABASE\_CONNECTION\_STRING”: jsii.String(connectionString), // Using the port token as part of the string }, })

// Output the value of our connection string at synthesis fmt.Println(“connectionString:”, connectionString)

// Output the connection string awscdk.NewCfnOutput(stack, jsii.String(“ConnectionString”), &awscdk.CfnOutputProps{ Value: jsii.String(connectionString), })

return stack } ….

== // …

====

If we pass this value to connectionString, the output value when we run cdk synth may be confusing due to the number-encoded string:

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

$ cdk synth –quiet connectionString: jdbc:mysql://mydb.cluster.amazonaws.com:-1.888154589708796e+289/mydatabase —

To convert a number-encoded token to a string, use https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tokenization.html#static-stringifywbrnumberx[cdk.Tokenization.stringifyNumber(<token>)]. In the following example, we convert the number-encoded token to a string before defining our connection string:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { Stack, Duration, Tokenization, CfnOutput, StackProps } from ‘aws-cdk-lib’; // …

export class CdkDemoAppStack extends Stack { constructor(scope: Construct, id: string, props?: StackProps) { super(scope, id, props);

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) const portToken = dbCluster.clusterEndpoint.port;

// …

// Convert the encoded number to an encoded string for use in the connection string const portAsString = Tokenization.stringifyNumber(portToken);

// Example connection string with the port token as a string const connectionString = jdbc:mysql://mydb.cluster.amazonaws.com:${portAsString}/mydatabase;

// Use the connection string as an environment variable in a Lambda function const myFunction = new lambda.Function(this, ‘MyLambdaFunction’, { // … environment: { DATABASE\_CONNECTION\_STRING: connectionString, // Using the port token as part of the string }, });

// Output the value of our connection string at synthesis console.log(“connectionString:” + connectionString);

// Output the connection string new CfnOutput(this, ‘ConnectionString’, { value: connectionString, }); } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { Stack, Duration, Tokenization, CfnOutput } = require(‘aws-cdk-lib’); // …

class CdkDemoAppStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) const portToken = dbCluster.clusterEndpoint.port;

// …

// Convert the encoded number to an encoded string for use in the connection string const portAsString = Tokenization.stringifyNumber(portToken);

// Example connection string with the port token as a string const connectionString = jdbc:mysql://mydb.cluster.amazonaws.com:${portAsString}/mydatabase;

// Use the connection string as an environment variable in a Lambda function const myFunction = new lambda.Function(this, ‘MyLambdaFunction’, { // … environment: { DATABASE\_CONNECTION\_STRING: connectionString, // Using the port token as part of the string }, });

// Output the value of our connection string at synthesis console.log(“connectionString:” + connectionString);

// Output the connection string new CfnOutput(this, ‘ConnectionString’, { value: connectionString, }); } } ….

== module.exports = { CdkDemoAppStack }

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( Duration, Stack, Tokenization, CfnOutput, )

= …

class CdkDemoAppStack(Stack):

…. def **init**(self, scope: Construct, construct\_id: str, \*\*kwargs) -> None: super().\_\_init\_\_(scope, construct\_id, \*\*kwargs)

# Define a new VPC  
# ...  
  
# Define an RDS database cluster  
# ...  
  
# Get the port token (this is a token encoded as a number)  
port\_token = db\_cluster.cluster\_endpoint.port  
  
# Convert the encoded number to an encoded string for use in the connection string  
port\_as\_string = Tokenization.stringify\_number(port\_token)  
  
# Example connection string with the port token as a string  
connection\_string = f"jdbc:mysql://mydb.cluster.amazonaws.com:{port\_as\_string}/mydatabase"  
  
# Use the connection string as an environment variable in a Lambda function  
my\_function = \_lambda.Function(self, 'MyLambdaFunction',  
 # ...  
 environment={  
 'DATABASE\_CONNECTION\_STRING': connection\_string # Using the port token as part of the string  
 }  
)  
  
# Output the value of our connection string at synthesis  
print(f"connectionString: {connection\_string}")  
  
# Output the connection string  
CfnOutput(self, 'ConnectionString',  
 value=connection\_string  
) ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — // … import software.amazon.awscdk.Tokenization;

public class CdkDemoAppStack extends Stack { public CdkDemoAppStack(final Construct scope, final String id) { this(scope, id, null); }

…. public CdkDemoAppStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// Define a new VPC  
// ...  
  
// Define an RDS database cluster  
// ...  
  
// Get the port token (this is a token encoded as a number)  
Number portToken = dbCluster.getClusterEndpoint().getPort();  
  
// ...  
  
// Convert the encoded number to an encoded string for use in the connection string  
String portAsString = Tokenization.stringifyNumber(portToken);  
  
// Example connection string with the port token as a string  
String connectionString = "jdbc:mysql://mydb.cluster.amazonaws.com:" + portAsString + "/mydatabase";  
  
// Use the connection string as an environment variable in a Lambda function  
Function myFunction = Function.Builder.create(this, "MyLambdaFunction")  
 // ...  
 .environment(Map.of(  
 "DATABASE\_CONNECTION\_STRING", connectionString // Using the port token as part of the string  
 ))  
 .build();  
  
// Output the value of our connection string at synthesis  
System.out.println("connectionString: " + connectionString);  
  
// Output the connection string  
CfnOutput.Builder.create(this, "ConnectionString")  
 .value(connectionString)  
 .build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — // …

namespace CdkDemoApp { public class CdkDemoAppStack : Stack { internal CdkDemoAppStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Define a new VPC // …

…. // Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number)  
 var portToken = dbCluster.ClusterEndpoint.Port;  
  
 // ...  
  
 // Convert the encoded number to an encoded string for use in the connection string  
 var portAsString = Tokenization.StringifyNumber(portToken);  
  
 // Example connection string with the port token as a string  
 var connectionString = $"jdbc:mysql://mydb.cluster.amazonaws.com:{portAsString}/mydatabase";  
  
 // Use the connection string as an environment variable in a Lambda function  
 var myFunction = new Function(this, "MyLambdaFunction", new FunctionProps  
 {  
 // ...  
 Environment = new Dictionary<string, string>  
 {  
 { "DATABASE\_CONNECTION\_STRING", connectionString } // Using the port token as part of the string  
 }  
 });  
  
 // Output the value of our connection string at synthesis  
 Console.WriteLine($"connectionString: {connectionString}");  
  
 // Output the connection string  
 new CfnOutput(this, "ConnectionString", new CfnOutputProps  
 {  
 Value = connectionString  
 });  
}

} } —- ….

Go:: + [source,go,subs=“verbatim,attributes”] — // …

func NewCdkDemoAppStack(scope constructs.Construct, id string, props \*CdkDemoAppStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Define a new VPC // …

// Define an RDS database cluster // …

// Get the port token (this is a token encoded as a number) portToken := dbCluster.ClusterEndpoint().Port()

// …

// Convert the encoded number to an encoded string for use in the connection string portAsString := awscdk.Tokenization\_StringifyNumber(portToken)

// Example connection string with the port token as a string connectionString := fmt.Sprintf(“jdbc:mysql://mydb.cluster.amazonaws.com:%s/mydatabase”, portAsString)

// Use the connection string as an environment variable in a Lambda function myFunction := awslambda.NewFunction(stack, jsii.String(“MyLambdaFunction”), &awslambda.FunctionProps{ // … Environment: &map[string]\*string{ “DATABASE\_CONNECTION\_STRING”: jsii.String(connectionString), // Using the port token as part of the string }, })

// Output the value of our connection string at synthesis fmt.Println(“connectionString:”, connectionString)

// Output the connection string awscdk.NewCfnOutput(stack, jsii.String(“ConnectionString”), &awscdk.CfnOutputProps{ Value: jsii.String(connectionString), })

fmt.Println(myFunction)

return stack } ….

== // …

====

When we run cdk synth, the value for our connection string is represented in a cleaner and clearer format:

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

$ cdk synth –quiet connectionString: jdbc:mysql://mydb.cluster.amazonaws.com:${Token[TOKEN.242]}/mydatabase —

[#tokens-lazy] == Lazy values

In addition to representing deploy-time values, such as {aws} CloudFormation xref:parameters[parameters], tokens are also commonly used to represent synthesis-time lazy values. These are values for which the final value will be determined before synthesis has completed, but not at the point where the value is constructed. Use tokens to pass a literal string or number value to another construct, while the actual value at synthesis time might depend on some calculation that has yet to occur.

You can construct tokens representing synth-time lazy values using static methods on the Lazy class, such as https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Lazy.html#static-stringproducer-options[Lazy.string] and https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Lazy.html#static-numberproducer[Lazy.number]. These methods accept an object whose produce property is a function that accepts a context argument and returns the final value when called.

The following example creates an Auto Scaling group whose capacity is determined after its creation.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — let actualValue: number;

new AutoScalingGroup(this, ‘Group’, { desiredCapacity: Lazy.numberValue({ produce(context) { return actualValue; } }) });

// At some later point actualValue = 10; —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — let actualValue;

new AutoScalingGroup(this, ‘Group’, { desiredCapacity: Lazy.numberValue({ produce(context) { return (actualValue); } }) });

// At some later point actualValue = 10; —

Python:: + [source,python,subs=“verbatim,attributes”] — class Producer: def *init*(self, func): self.produce = func

actual\_value = None

AutoScalingGroup(self, “Group”, desired\_capacity=Lazy.number\_value(Producer(lambda context: actual\_value)) )

= At some later point

## actual\_value = 10

Java:: + [source,java,subs=“verbatim,attributes”] — double actualValue = 0;

class ProduceActualValue implements INumberProducer {

@Override public Number produce(IResolveContext context) { return actualValue; } }

AutoScalingGroup.Builder.create(this, “Group”) .desiredCapacity(Lazy.numberValue(new ProduceActualValue())).build();

// At some later point actualValue = 10; —

C#:: + [source,csharp,subs=“verbatim,attributes”] — public class NumberProducer : INumberProducer { Func++++++function;++++++

…. public NumberProducer(Func function) { this.function = function; }

public Double Produce(IResolveContext context) { return function(); } } ….

double actualValue = 0;

new AutoScalingGroup(this, “Group”, new AutoScalingGroupProps { DesiredCapacity = Lazy.NumberValue(new NumberProducer(() => actualValue)) });

// At some later point actualValue = 10; — ====

[#tokens-json] == Converting to JSON

Sometimes you want to produce a JSON string of arbitrary data, and you may not know whether the data contains tokens. To properly JSON-encode any data structure, regardless of whether it contains tokens, use the method https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#towbrjsonwbrstringobj-space[stack.toJsonString], as shown in the following example.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const stack = Stack.of(this); const str = stack.toJsonString({ value: bucket.bucketName }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const stack = Stack.of(this); const str = stack.toJsonString({ value: bucket.bucketName }); —

Python:: + [source,python,subs=“verbatim,attributes”] — stack = Stack.of(self) string = stack.to\_json\_string(dict(value=bucket.bucket\_name)) —

Java:: + [source,java,subs=“verbatim,attributes”] — Stack stack = Stack.of(this); String stringVal = stack.toJsonString(java.util.Map.of( // Map.of requires Java 9+ put(“value”, bucket.getBucketName()))); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var stack = Stack.Of(this); var stringVal = stack.ToJsonString(new Dictionary<string, string> { [“value”] = bucket.BucketName }); — ====