:doctype: book

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Context values :keywords: {aws} CDK, Context values

[#context] = Context values and the {aws} CDK

== [abstract]

## Context values are key-value pairs that can be associated with an app, stack, or construct. They may be supplied to your app from a file (usually either cdk.json or cdk.context.json in your project directory) or on the command line.

// Context start

Context values are key-value pairs that can be associated with an app, stack, or construct. They may be supplied to your app from a file (usually either cdk.json or cdk.context.json in your project directory) or on the command line.

The CDK Toolkit uses context to cache values retrieved from your {aws} account during synthesis. Values include the Availability Zones in your account or the Amazon Machine Image (AMI) IDs currently available for Amazon EC2 instances. Because these values are provided by your {aws} account, they can change between runs of your CDK application. This makes them a potential source of unintended change. The CDK Toolkit’s caching behavior “freezes” these values for your CDK app until you decide to accept the new values.

Imagine the following scenario without context caching. Let’s say you specified “latest Amazon Linux” as the AMI for your Amazon EC2 instances, and a new version of this AMI was released. Then, the next time you deployed your CDK stack, your already-deployed instances would be using the outdated (“wrong”) AMI and would need to be upgraded. Upgrading would result in replacing all your existing instances with new ones, which would probably be unexpected and undesired.

Instead, the CDK records your account’s available AMIs in your project’s cdk.context.json file, and uses the stored value for future synthesis operations. This way, the list of AMIs is no longer a potential source of change. You can also be sure that your stacks will always synthesize to the same {aws} CloudFormation templates.

Not all context values are cached values from your {aws} environment. xref:featureflags[{aws} CDK feature flags] are also context values. You can also create your own context values for use by your apps or constructs.

Context keys are strings. Values may be any type supported by JSON: numbers, strings, arrays, or objects.

= [TIP]

If your constructs create their own context values, incorporate your library’s package name in its keys so they won’t conflict with other packages’ context values.

====

Many context values are associated with a particular {aws} environment, and a given CDK app can be deployed in more than one environment. The key for such values includes the {aws} account and Region so that values from different environments do not conflict.

The following context key illustrates the format used by the {aws} CDK, including the account and Region.

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

## availability-zones:account=123456789012:region=eu-central-1

= [IMPORTANT]

Cached context values are managed by the {aws} CDK and its constructs, including constructs you may write. Do not add or change cached context values by manually editing files. It can be useful, however, to review cdk.context.json occasionally to see what values are being cached. Context values that don’t represent cached values should be stored under the context key of cdk.json. This way, they won’t be cleared when cached values are cleared.

====

[#context-construct] == Sources of context values

Context values can be provided to your {aws} CDK app in six different ways:

* Automatically from the current {aws} account.
* Through the --context option to the cdk command. (These values are always strings.)
* In the project’s cdk.context.json file.
* In the context key of the project’s cdk.json file.
* In the context key of your ~/.cdk.json file.
* In your {aws} CDK app using the construct.node.setContext() method.

The project file cdk.context.json is where the {aws} CDK caches context values retrieved from your {aws} account. This practice avoids unexpected changes to your deployments when, for example, a new Availability Zone is introduced. The {aws} CDK does not write context data to any of the other files listed.

= [IMPORTANT]

Because they’re part of your application’s state, cdk.json and cdk.context.json must be committed to source control along with the rest of your app’s source code. Otherwise, deployments in other environments (for example, a CI pipeline) might produce inconsistent results.

====

Context values are scoped to the construct that created them; they are visible to child constructs, but not to parents or siblings. Context values that are set by the {aws} CDK Toolkit (the cdk command) can be set automatically, from a file, or from the --context option. Context values from these sources are implicitly set on the App construct. Therefore, they’re visible to every construct in every stack in the app.

Your app can read a context value using the construct.node.tryGetContext method. If the requested entry isn’t found on the current construct or any of its parents, the result is undefined. (Alternatively, the result could be your language’s equivalent, such as None in Python.)

[#context-methods] == Context methods

The {aws} CDK supports several context methods that enable {aws} CDK apps to obtain contextual information from the {aws} environment. For example, you can get a list of Availability Zones that are available in a given {aws} account and Region, using the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#availabilityzones[stack.availabilityZones] method.

The following are the context methods:

link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_route53.HostedZone.html#static-fromwbrlookupscope-id-query[HostedZone.fromLookup]:: + Gets the hosted zones in your account.

link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#availabilityzones[stack.availabilityZones]:: + Gets the supported Availability Zones.

link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ssm.StringParameter.html#static-valuewbrfromwbrlookupscope-parametername[StringParameter.valueFromLookup]:: + Gets a value from the current Region’s Amazon EC2 Systems Manager Parameter Store.

link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ec2.Vpc.html#static-fromwbrlookupscope-id-options[Vpc.fromLookup]:: + Gets the existing Amazon Virtual Private Clouds in your accounts.

link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ec2.LookupMachineImage.html[LookupMachineImage]:: + Looks up a machine image for use with a NAT instance in an Amazon Virtual Private Cloud.

If a required context value isn’t available, the {aws} CDK app notifies the CDK Toolkit that the context information is missing. Next, the CLI queries the current {aws} account for the information and stores the resulting context information in the cdk.context.json file. Then, it executes the {aws} CDK app again with the context values.

[#context-viewing] == Viewing and managing context

Use the cdk context command to view and manage the information in your cdk.context.json file. To see this information, use the cdk context command without any options. The output should be something like the following.

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

Context found in cdk.json:

┌───┬─────────────────────────────────────────────────────────────┬─────────────────────────────────────────────────────────┐ │ # │ Key │ Value │ ├───┼─────────────────────────────────────────────────────────────┼─────────────────────────────────────────────────────────┤ │ 1 │ availability-zones:account=123456789012:region=eu-central-1 │ [ “eu-central-1a”, “eu-central-1b”, “eu-central-1c” ] │ ├───┼─────────────────────────────────────────────────────────────┼─────────────────────────────────────────────────────────┤ │ 2 │ availability-zones:account=123456789012:region=eu-west-1 │ [ “eu-west-1a”, “eu-west-1b”, “eu-west-1c” ] │ └───┴─────────────────────────────────────────────────────────────┴─────────────────────────────────────────────────────────┘

Run

cdk context –reset KEY\_OR\_NUMBER

to remove a context key. If it is a cached value, it will be refreshed on the next

cdk synth

== .

To remove a context value, run cdk context --reset, specifying the value’s corresponding key or number. The following example removes the value that corresponds to the second key in the preceding example. This value represents the list of Availability Zones in the Europe (Ireland) Region.

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

## cdk context –reset 2

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

Context value availability-zones:account=123456789012:region=eu-west-1 reset. It will be refreshed on the next SDK synthesis run. —

Therefore, if you want to update to the latest version of the Amazon Linux AMI, use the preceding example to do a controlled update of the context value and reset it. Then, synthesize and deploy your app again.

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

## $ cdk synth

To clear all of the stored context values for your app, run cdk context --clear, as follows.

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

## $ cdk context –clear

Only context values stored in cdk.context.json can be reset or cleared. The {aws} CDK does not touch other context values. Therefore, to protect a context value from being reset using these commands, you might copy the value to cdk.json.

[#context-cli] == {aws} CDK Toolkit --context flag

Use the --context (-c for short) option to pass runtime context values to your CDK app during synthesis or deployment.

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

## $ cdk synth –context key=value MyStack

To specify multiple context values, repeat the --context option any number of times, providing one key-value pair each time.

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

## $ cdk synth –context key1=value1 –context key2=value2 MyStack

When synthesizing multiple stacks, the specified context values are passed to all stacks. To provide different context values to individual stacks, either use different keys for the values, or use multiple cdk synth or cdk deploy commands.

Context values passed from the command line are always strings. If a value is usually of some other type, your code must be prepared to convert or parse the value. You might have non-string context values provided in other ways (for example, in cdk.context.json). To make sure this kind of value works as expected, confirm that the value is a string before converting it.

[#context-example] == Example

Following is an example of using an existing Amazon VPC using {aws} CDK context.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import \* as ec2 from ‘aws-cdk-lib/aws-ec2’; import { Construct } from ‘constructs’;

export class ExistsVpcStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

…. super(scope, id, props);

const vpcid = this.node.tryGetContext(‘vpcid’); const vpc = ec2.Vpc.fromLookup(this, ‘VPC’, { vpcId: vpcid, });

const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

new cdk.CfnOutput(this, ‘publicsubnets’, { value: pubsubnets.subnetIds.toString(), }); } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const cdk = require(‘aws-cdk-lib’); const ec2 = require(‘aws-cdk-lib/aws-ec2’);

class ExistsVpcStack extends cdk.Stack {

constructor(scope, id, props) {

…. super(scope, id, props);

const vpcid = this.node.tryGetContext(‘vpcid’); const vpc = ec2.Vpc.fromLookup(this, ‘VPC’, { vpcId: vpcid });

const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

new cdk.CfnOutput(this, ‘publicsubnets’, { value: pubsubnets.subnetIds.toString() }); } } ….

== module.exports = { ExistsVpcStack }

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk import aws\_cdk.aws\_ec2 as ec2 from constructs import Construct

class ExistsVpcStack(cdk.Stack):

…. def **init**(scope: Construct, id: str, \*\*kwargs):

super().\_\_init\_\_(scope, id, \*\*kwargs)  
  
vpcid = self.node.try\_get\_context("vpcid")  
vpc = ec2.Vpc.from\_lookup(self, "VPC", vpc\_id=vpcid)  
  
pubsubnets = vpc.select\_subnets(subnetType=ec2.SubnetType.PUBLIC)  
  
cdk.CfnOutput(self, "publicsubnets",  
 value=pubsubnets.subnet\_ids.to\_string()) ----

….

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

import software.amazon.awscdk.services.ec2.Vpc; import software.amazon.awscdk.services.ec2.VpcLookupOptions; import software.amazon.awscdk.services.ec2.SelectedSubnets; import software.amazon.awscdk.services.ec2.SubnetSelection; import software.amazon.awscdk.services.ec2.SubnetType; import software.constructs.Construct;

public class ExistsVpcStack extends Stack { public ExistsVpcStack(Construct context, String id) { this(context, id, null); }

…. public ExistsVpcStack(Construct context, String id, StackProps props) { super(context, id, props);

String vpcId = (String)this.getNode().tryGetContext("vpcid");  
Vpc vpc = (Vpc)Vpc.fromLookup(this, "VPC", VpcLookupOptions.builder()  
 .vpcId(vpcId).build());  
  
SelectedSubnets pubSubNets = vpc.selectSubnets(SubnetSelection.builder()  
 .subnetType(SubnetType.PUBLIC).build());  
  
CfnOutput.Builder.create(this, "publicsubnets")  
 .value(pubSubNets.getSubnetIds().toString()).build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Amazon.CDK.{aws}.EC2; using Constructs;

class ExistsVpcStack : Stack { public ExistsVpcStack(Construct scope, string id, StackProps props) : base(scope, id, props) { var vpcId = (string)this.Node.TryGetContext(“vpcid”); var vpc = Vpc.FromLookup(this, “VPC”, new VpcLookupOptions { VpcId = vpcId });

…. SelectedSubnets pubSubNets = vpc.SelectSubnets([new SubnetSelection { SubnetType = SubnetType.PUBLIC }]);

new CfnOutput(this, "publicsubnets", new CfnOutputProps {  
 Value = pubSubNets.SubnetIds.ToString()  
});

} } —- ==== ….

You can use cdk diff to see the effects of passing in a context value on the command line:

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

## $ cdk diff -c vpcid=vpc-0cb9c31031d0d3e22

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

Stack ExistsvpcStack Outputs [+] Output publicsubnets publicsubnets: {“Value”:“subnet-06e0ea7dd302d3e8f,subnet-01fc0acfb58f3128f”} —

The resulting context values can be viewed as shown here.

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

## $ cdk context -j

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

{ “vpc-provider:account=123456789012:filter.vpc-id=vpc-0cb9c31031d0d3e22:region=us-east-1”: { “vpcId”: “vpc-0cb9c31031d0d3e22”, “availabilityZones”: [ “us-east-1a”, “us-east-1b” ], “privateSubnetIds”: [ “subnet-03ecfc033225be285”, “subnet-0cded5da53180ebfa” ], “privateSubnetNames”: [ “Private” ], “privateSubnetRouteTableIds”: [ “rtb-0e955393ced0ada04”, “rtb-05602e7b9f310e5b0” ], “publicSubnetIds”: [ “subnet-06e0ea7dd302d3e8f”, “subnet-01fc0acfb58f3128f” ], “publicSubnetNames”: [ “Public” ], “publicSubnetRouteTableIds”: [ “rtb-00d1fdfd823c82289”, “rtb-04bb1969b42969bcb” ] } } —