:doctype: book

include::attributes.txt[]

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

[#permissions] = Permissions and the {aws} CDK

== [abstract]

## The {aws} Construct Library uses a few common, widely implemented idioms to manage access and permissions. The IAM module provides you with the tools you need to use these idioms.

// Content start

The {aws} Construct Library uses a few common, widely implemented idioms to manage access and permissions. The IAM module provides you with the tools you need to use these idioms.

{aws} CDK uses {aws} CloudFormation to deploy changes. Every deployment involves an actor (either a developer, or an automated system) that starts a {aws} CloudFormation deployment. In the course of doing this, the actor will assume one or more IAM Identities (user or roles) and optionally pass a role to {aws} CloudFormation.

If you use {aws} IAM Identity Center to authenticate as a user, then the single sign-on provider supplies short-lived session credentials that authorize you to act as a pre-defined IAM role. To learn how the {aws} CDK obtains {aws} credentials from IAM Identity Center authentication, see https://docs.aws.amazon.com/sdkref/latest/guide/understanding-sso.html[Understand IAM Identity Center authentication] in the *{aws} SDKs and Tools Reference Guide*.

[#permissions-principals] == Principals

An IAM principal is an authenticated {aws} entity representing a user, service, or application that can call {aws} APIs. The {aws} Construct Library supports specifying principals in several flexible ways to grant them access your {aws} resources.

In security contexts, the term “principal” refers specifically to authenticated entities such as users. Objects like groups and roles do not *represent* users (and other authenticated entities) but rather *identify* them indirectly for the purpose of granting permissions.

For example, if you create an IAM group, you can grant the group (and thus its members) write access to an Amazon RDS table. However, the group itself is not a principal because it doesn’t represent a single entity (also, you cannot log in to a group).

In the CDK’s IAM library, classes that directly or indirectly identify principals implement the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.IPrincipal.html[IPrincipal] interface, allowing these objects to be used interchangeably in access policies. However, not all of them are principals in the security sense. These objects include:

. IAM resources such as https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role], https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html[User], and https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html[Group] . Service principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ServicePrincipal.html[ServicePrincipal]('service.amazonaws.com')+) . Federated principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.FederatedPrincipal.html[FederatedPrincipal]('cognito-identity.amazonaws.com')+) . Account principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.AccountPrincipal.html[AccountPrincipal]('0123456789012')+) . Canonical user principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.CanonicalUserPrincipal.html[CanonicalUserPrincipal]('79a59d[...]7ef2be')+) . {aws} Organizations principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.OrganizationPrincipal.html[OrganizationPrincipal]('org-id')+) . Arbitrary ARN principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ArnPrincipal.html[ArnPrincipal](res.arn)+) . An +iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.CompositePrincipal.html[CompositePrincipal](principal1, principal2, ...)+ to trust multiple principals

[#permissions-grants] == Grants

Every construct that represents a resource that can be accessed, such as an Amazon S3 bucket or Amazon DynamoDB table, has methods that grant access to another entity. All such methods have names starting with *grant*.

For example, Amazon S3 buckets have the methods link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#grantwbrreadidentity-objectskeypattern[grantRead] and link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#grantwbrreadwbrwriteidentity-objectskeypattern[grantReadWrite] (Python: grant\_read, grant\_read\_write) to enable read and read/write access, respectively, from an entity to the bucket. The entity doesn’t have to know exactly which Amazon S3 IAM permissions are required to perform these operations.

The first argument of a *grant* method is always of type link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.IGrantable.html[IGrantable]. This interface represents entities that can be granted permissions. That is, it represents resources with roles, such as the IAM objects link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role], link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html[User], and link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html[Group].

Other entities can also be granted permissions. For example, later in this topic, we show how to grant a CodeBuild project access to an Amazon S3 bucket. Generally, the associated role is obtained via a role property on the entity being granted access.

Resources that use execution roles, such as link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Function.html[lambda.Function], also implement IGrantable, so you can grant them access directly instead of granting access to their role. For example, if bucket is an Amazon S3 bucket, and function is a Lambda function, the following code grants the function read access to the bucket.

==== [role=“tablist”] TypeScript:: + [source,none,subs=“verbatim,attributes”] — bucket.grantRead(function); —

JavaScript:: + [source,none,subs=“verbatim,attributes”] — bucket.grantRead(function); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket.grant\_read(function) —

Java:: + [source,java,subs=“verbatim,attributes”] — bucket.grantRead(function); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — bucket.GrantRead(function); — ====

Sometimes permissions must be applied while your stack is being deployed. One such case is when you grant an {aws} CloudFormation custom resource access to some other resource. The custom resource will be invoked during deployment, so it must have the specified permissions at deployment time.

Another case is when a service verifies that the role you pass to it has the right policies applied. (A number of {aws} services do this to make sure that you didn’t forget to set the policies.) In those cases, the deployment might fail if the permissions are applied too late.

To force the grant’s permissions to be applied before another resource is created, you can add a dependency on the grant itself, as shown here. Though the return value of grant methods is commonly discarded, every grant method in fact returns an iam.Grant object.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const grant = bucket.grantRead(lambda); const custom = new CustomResource(…); custom.node.addDependency(grant); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const grant = bucket.grantRead(lambda); const custom = new CustomResource(…); custom.node.addDependency(grant); —

Python:: + [source,python,subs=“verbatim,attributes”] — grant = bucket.grant\_read(function) custom = CustomResource(…) custom.node.add\_dependency(grant) —

Java:: + [source,java,subs=“verbatim,attributes”] — Grant grant = bucket.grantRead(function); CustomResource custom = new CustomResource(…); custom.node.addDependency(grant); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var grant = bucket.GrantRead(function); var custom = new CustomResource(…); custom.node.AddDependency(grant); — ====

[#permissions-roles] == Roles

The IAM package contains a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role] construct that represents IAM roles. The following code creates a new role, trusting the Amazon EC2 service.

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

const role = new iam.Role(this, ‘Role’, { assumedBy: new iam.ServicePrincipal(‘ec2.amazonaws.com’), // required }); —

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

const role = new iam.Role(this, ‘Role’, { assumedBy: new iam.ServicePrincipal(‘ec2.amazonaws.com’) // required }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_iam as iam

role = iam.Role(self, “Role”, assumed\_by=iam.ServicePrincipal(“ec2.amazonaws.com”)) # required —

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

Role role = Role.Builder.create(this, “Role”) .assumedBy(new ServicePrincipal(“ec2.amazonaws.com”)).build(); —

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

var role = new Role(this, “Role”, new RoleProps { AssumedBy = new ServicePrincipal(“ec2.amazonaws.com”), // required }); — ====

You can add permissions to a role by calling the role’s link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#addwbrtowbrpolicystatement[addToPolicy] method (Python: add\_to\_policy), passing in a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] that defines the rule to be added. The statement is added to the role’s default policy; if it has none, one is created.

The following example adds a Deny policy statement to the role for the actions ec2:SomeAction and s3:AnotherAction on the resources bucket and otherRole (Python: other\_role), under the condition that the authorized service is {aws} CodeBuild.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — role.addToPolicy(new iam.PolicyStatement({ effect: iam.Effect.DENY, resources: [bucket.bucketArn, otherRole.roleArn], actions: [‘ec2:SomeAction’, ‘s3:AnotherAction’], conditions: {StringEquals: { ‘ec2:AuthorizedService’: ‘codebuild.amazonaws.com’, }}})); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — role.addToPolicy(new iam.PolicyStatement({ effect: iam.Effect.DENY, resources: [bucket.bucketArn, otherRole.roleArn], actions: [‘ec2:SomeAction’, ‘s3:AnotherAction’], conditions: {StringEquals: { ‘ec2:AuthorizedService’: ‘codebuild.amazonaws.com’ }}})); —

Python:: + [source,python,subs=“verbatim,attributes”] — role.add\_to\_policy(iam.PolicyStatement( effect=iam.Effect.DENY, resources=[bucket.bucket\_arn, other\_role.role\_arn], actions=[“ec2:SomeAction”, “s3:AnotherAction”], conditions={“StringEquals”: { “ec2:AuthorizedService”: “codebuild.amazonaws.com”}} )) —

Java:: + [source,java,subs=“verbatim,attributes”] — role.addToPolicy(PolicyStatement.Builder.create() .effect(Effect.DENY) .resources(Arrays.asList(bucket.getBucketArn(), otherRole.getRoleArn())) .actions(Arrays.asList(“ec2:SomeAction”, “s3:AnotherAction”)) .conditions(java.util.Map.of( // Map.of requires Java 9 or later “StringEquals”, java.util.Map.of( “ec2:AuthorizedService”, “codebuild.amazonaws.com”))) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — role.AddToPolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.DENY, Resources = new string[] { bucket.BucketArn, otherRole.RoleArn }, Actions = new string[] { “ec2:SomeAction”, “s3:AnotherAction” }, Conditions = new Dictionary<string, object> { [“StringEquals”] = new Dictionary<string, string> { [“ec2:AuthorizedService”] = “codebuild.amazonaws.com” } } })); — ====

In the preceding example, we’ve created a new link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] inline with the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#addwbrtowbrpolicystatement[addToPolicy] (Python: add\_to\_policy) call. You can also pass in an existing policy statement or one you’ve modified. The link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] object has link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html#methods[numerous methods] for adding principals, resources, conditions, and actions.

If you’re using a construct that requires a role to function correctly, you can do one of the following:

* Pass in an existing role when instantiating the construct object.
* Let the construct create a new role for you, trusting the appropriate service principal. The following example uses such a construct: a CodeBuild project.

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

// imagine roleOrUndefined is a function that might return a Role object // under some conditions, and undefined under other conditions const someRole: iam.IRole | undefined = roleOrUndefined();

const project = new codebuild.Project(this, ‘Project’, { // if someRole is undefined, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal role: someRole, }); —

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

// imagine roleOrUndefined is a function that might return a Role object // under some conditions, and undefined under other conditions const someRole = roleOrUndefined();

const project = new codebuild.Project(this, ‘Project’, { // if someRole is undefined, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal role: someRole }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_codebuild as codebuild

= imagine role\_or\_none is a function that might return a Role object

= under some conditions, and None under other conditions

some\_role = role\_or\_none();

project = codebuild.Project(self, “Project”,

= if role is None, the Project creates a new default role,

= trusting the codebuild.amazonaws.com service principal

## role=some\_role)

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.iam.Role; import software.amazon.awscdk.services.codebuild.Project;

// imagine roleOrNull is a function that might return a Role object // under some conditions, and null under other conditions Role someRole = roleOrNull();

// if someRole is null, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal Project project = Project.Builder.create(this, “Project”) .role(someRole).build(); —

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

// imagine roleOrNull is a function that might return a Role object // under some conditions, and null under other conditions var someRole = roleOrNull();

// if someRole is null, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal var project = new Project(this, “Project”, new ProjectProps { Role = someRole }); — ====

Once the object is created, the role (whether the role passed in or the default one created by the construct) is available as the property role. However, this property is not available on external resources. Therefore, these constructs have an addToRolePolicy (Python: add\_to\_role\_policy) method.

The method does nothing if the construct is an external resource, and it calls the addToPolicy (Python: add\_to\_policy) method of the role property otherwise. This saves you the trouble of handling the undefined case explicitly.

The following example demonstrates:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // project is imported into the CDK application const project = codebuild.Project.fromProjectName(this, ‘Project’, ‘ProjectName’);

// project is imported, so project.role is undefined, and this call has no effect project.addToRolePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, // … and so on defining the policy })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // project is imported into the CDK application const project = codebuild.Project.fromProjectName(this, ‘Project’, ‘ProjectName’);

// project is imported, so project.role is undefined, and this call has no effect project.addToRolePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW // … and so on defining the policy })); —

Python:: + [source,python,subs=“verbatim,attributes”] —

= project is imported into the CDK application

project = codebuild.Project.from\_project\_name(self, ‘Project’, ‘ProjectName’)

= project is imported, so project.role is undefined, and this call has no effect

project.add\_to\_role\_policy(iam.PolicyStatement( effect=iam.Effect.ALLOW, # … and so on defining the policy ) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — // project is imported into the CDK application Project project = Project.fromProjectName(this, “Project”, “ProjectName”);

// project is imported, so project.getRole() is null, and this call has no effect project.addToRolePolicy(PolicyStatement.Builder.create() .effect(Effect.ALLOW) // .. and so on defining the policy .build(); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // project is imported into the CDK application var project = Project.FromProjectName(this, “Project”, “ProjectName”);

// project is imported, so project.role is null, and this call has no effect project.AddToRolePolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.ALLOW, // … and so on defining the policy })); — ====

[#permissions-resource-policies] == Resource policies

A few resources in {aws}, such as Amazon S3 buckets and IAM roles, also have a resource policy. These constructs have an addToResourcePolicy method (Python: add\_to\_resource\_policy), which takes a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] as its argument. Every policy statement added to a resource policy must specify at least one principal.

In the following example, the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[Amazon S3 bucket] bucket grants a role with the s3:SomeAction permission to itself.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, actions: [‘s3:SomeAction’], resources: [bucket.bucketArn], principals: [role] })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, actions: [‘s3:SomeAction’], resources: [bucket.bucketArn], principals: [role] })); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket.add\_to\_resource\_policy(iam.PolicyStatement( effect=iam.Effect.ALLOW, actions=[“s3:SomeAction”], resources=[bucket.bucket\_arn], principals=role)) —

Java:: + [source,java,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(PolicyStatement.Builder.create() .effect(Effect.ALLOW) .actions(Arrays.asList(“s3:SomeAction”)) .resources(Arrays.asList(bucket.getBucketArn())) .principals(Arrays.asList(role)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — bucket.AddToResourcePolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.ALLOW, Actions = new string[] { “s3:SomeAction” }, Resources = new string[] { bucket.BucketArn }, Principals = new IPrincipal[] { role } })); — ====

[#permissions-existing] == Using external IAM objects

If you have defined an IAM user, principal, group, or role outside your {aws} CDK app, you can use that IAM object in your {aws} CDK app. To do so, create a reference to it using its ARN or its name. (Use the name for users, groups, and roles.) The returned reference can then be used to grant permissions or to construct policy statements as explained previously.

* For users, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html#static-fromwbruserwbrarnscope-id-userarn[User.fromUserArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html#static-fromwbruserwbrnamescope-id-username[User.fromUserName()]. User.fromUserAttributes() is also available, but currently provides the same functionality as User.fromUserArn().
* For principals, instantiate an link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ArnPrincipal.html[ArnPrincipal] object.
* For groups, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html#static-fromwbrgroupwbrarnscope-id-grouparn[Group.fromGroupArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html#static-fromwbrgroupwbrnamescope-id-groupname[Group.fromGroupName()].
* For roles, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#static-fromwbrrolewbrarnscope-id-rolearn-options[Role.fromRoleArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#static-fromwbrrolewbrnamescope-id-rolename[Role.fromRoleName()].

Policies (including managed policies) can be used in similar fashion using the following methods. You can use references to these objects anywhere an IAM policy is required.

* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Policy.html#static-fromwbrpolicywbrnamescope-id-policyname[Policy.fromPolicyName]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrmanagedwbrpolicywbrarnscope-id-managedpolicyarn[ManagedPolicy.fromManagedPolicyArn]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrmanagedwbrpolicywbrnamescope-id-managedpolicyname[ManagedPolicy.fromManagedPolicyName]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrawswbrmanagedwbrpolicywbrnamemanagedpolicyname[ManagedPolicy.fromAwsManagedPolicyName]

= [NOTE]

As with all references to external {aws} resources, you cannot modify external IAM objects in your CDK app.

====