Amazon Cloud Directory

Developer Guide

Amazon Cloud Directory: Developer Guide

Table of Contents

What Is Amazon Cloud Directory?

Amazon Cloud Directory is a highly available multi-tenant directory-based store in AWS. These directories scale automatically to hundreds of millions of objects as needed for applications. This lets operation's staff focus on developing and deploying applications that drive the business, not managing directory infrastructure. Unlike traditional directory systems, Cloud Directory does not limit organizing directory objects in a single fixed hierarchy.

With Cloud Directory, you can organize directory objects into multiple hierarchies to support many organizational pivots and relationships across directory information. For example, a directory of users may provide a hierarchical view based on reporting structure, location, and project affiliation. Similarly, a directory of devices may have multiple hierarchical views based on its manufacturer, current owner, and physical location.

At its core, Cloud Directory is a specialized graph-based directory store that provides a foundational building block for developers. With Cloud Directory, developers can do the following:

- Create directory-based applications easily and without having to worry about deployment, global scale, availability, and performance
- Build applications that provide user and group management, permissions or policy management, device registry, customer management, address books, and application or product catalogs
- Define new directory objects or extend existing types to meet their application needs, reducing the code they need to write
- Reduce the complexity of layering applications on top of Cloud Directory
- Manage the evolution of schema information over time, ensuring future compatibility for consumers

Cloud Directory includes a set of API operations to access various objects and policies stored in your Cloud Directory-based directories. For a list of available operations, see Amazon Cloud Directory API Actions. For a list of operations and the permissions required to perform each API action, see Amazon Cloud Directory API Permissions: Actions, Resources, and Conditions Reference (p. 53).

For a list of supported Cloud Directory regions, see the AWS Regions and Endpoints documentation. For additional resources, see Cloud Directory Resources (p. 66).

What Cloud Directory Is Not

Cloud Directory is not a directory service for IT Administrators who want to manage or migrate their directory infrastructure.

Getting Started

In this getting started exercise, you create a schema. You then choose to create a directory from that same schema or from any of the sample schemas that are available in the AWS Directory Service console. Although not required, we recommend that you review Understanding Key Cloud Directory Concepts (p. 4) before you begin using the console so that you are familiar with the core features and terminology.

Topics

Create a Schema

Amazon Cloud Directory supports uploading of a compliant JSON file for schema creation. To create a new schema, you can either create your own JSON file from scratch or download one of the existing schemas listed in the console. Then upload it as a custom schema. For more information, see Custom Schemas (p. 17).

You can also create, delete, download, list, publish, update and upgrade schemas using the Cloud Directory APIs. For more information about schema API operations, see the Amazon Cloud Directory API Reference Guide.

Choose either of the procedures below, depending on your preferred method.

To create a custom schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Schemas**.
2. Create a JSON file with all of your new schema definitions. For more information about how to format a JSON file, see JSON Schema Format (p. 21).
3. In the console, choose **Upload new schema**.
4. In the **Upload new schema** dialog, type a name for the schema.
5. Select **Choose file**, select the new JSON file that you just created, and then choose **Open**.
6. Choose **Upload**. This adds a new schema to your schema library and places it in the **Development** state. For more information about schema states, see Schema Lifecycle (p. 8).

To create a custom schema based on an existing one in the console

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Schemas**.
2. In the table listing the schemas, select the option near the schema you want to copy.
3. Choose **Actions**.
4. Choose **Download schema**.
5. Rename the JSON file, edit it as needed, and then save the file. For more information about how to format a JSON file, see JSON Schema Format (p. 21).
6. In the console, choose **Upload new schema**, select the JSON file that you just edited, and then choose **Open**.

   This adds a new schema to your schema library and places it in the **Development** state. For more information about schema states, see Schema Lifecycle (p. 8).

Create an Amazon Cloud Directory

Before you can create a directory in Amazon Cloud Directory, AWS Directory Service requires that you first apply a schema to it. A directory cannot be created without a schema and typically has one schema applied to it. However, you use Cloud Directory API operations to apply additional schemas to a directory. For more information, see ApplySchema in the *Amazon Cloud Directory API Reference Guide*.

To create a Cloud Directory

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Directories**.
2. Choose **Set up Cloud Directory**.
3. Under **Choose a schema to apply to your new directory**, type the friendly name of your directory, such as `User Repository`, and then choose one of the following options:

   - **Managed schema**
   - **Sample schema**
   - **Custom schema**

   Sample schemas and custom schemas are placed in the **Development** state, by default. For more information about schema states, see Schema Lifecycle (p. 8). Before a schema can be applied to a directory, it must be converted into the **Published** state. To successfully publish a sample schema using the console, you must have permissions to the following actions:

   - `clouddirectory:Get*`
   - `clouddirectory:List*`
   - `clouddirectory:CreateSchema`
   - `clouddirectory:CreateDirectory`
   - `clouddirectory:PutSchemaFromJson`
   - `clouddirectory:PublishSchema`
   - `clouddirectory:DeleteSchema`

   Since sample schemas are read-only templates provided by AWS, they cannot be published directly. Instead, when you choose to create a directory based on a sample schema, the console creates a temporary copy of the sample schema you selected and places it in the **Development** state. It then creates a copy of that development schema and places it in the **Published** state. Once published, the development schema is deleted, which is why the `DeleteSchema` action is necessary when publishing a sample schema.
4. Choose **Next**.
5. Review the directory information and make any necessary changes. When the information is correct, choose **Create**.

Understanding Key Cloud Directory Concepts

Amazon Cloud Directory is a directory-based data store that can create various types of objects in a schema-oriented fashion.

Topics

Schema

A schema is a collection of facets that define what objects can be created in a directory and how they are organized. A schema also enforces data integrity and interoperability. A single schema can be applied to more than one directory at a time. For more information, see Schemas (p. 8).

Facets

A facet is a collection of attributes, constraints, and links defined within a schema. Combined together, facets define the objects in a directory. For example, Person and Device can be facets to define corporate employees with association of multiple devices. For more information, see Facets (p. 10).

Managed Schemas

A schema provided to make it easier to quickly develop and maintain your applications. For more information, see Managed Schema (p. 11).

Sample Schemas

The set of sample schemas provided by default in the AWS Directory Service console. For example, Person, Organization, and Device are all sample schemas. For more information, see Sample Schemas (p. 13).

Custom Schemas

One or more schemas defined by a user that can be uploaded from the Schemas section or during the Cloud Directory creation process of the AWS Directory Service console, or created by API calls.

Directory

A directory is a schema-based data store that contains specific types of objects organized in a multi-hierarchical structure (see Directory Structure (p. 6) for more details). For example, a directory of users may provide a hierarchical view based on reporting structure, location, and project affiliation.

Similarly, a directory of devices may have multiple hierarchical views based on its manufacturer, current owner, and physical location.

A directory defines the logical boundary for the data store, completely isolating it from all other directories in the service. It also defines the boundaries for an individual request. A single transaction or query executes within the context of a single directory. A directory cannot be created without a schema and typically has one schema applied to it. However, you can use the Cloud Directory API operations to apply additional schemas to a directory. For more information, see `ApplySchema` in the *Amazon Cloud Directory API Reference Guide*.

Objects

Objects are a structured data entity in a directory. An object in a directory is intended to capture metadata (or attributes) about a physical or logical entity usually for the purpose of information discovery and enforcing policies. For example users, devices, applications, AWS accounts, EC2 instances and Amazon S3 buckets can all be represented as different types of objects in a directory.

An object's structure and type information is expressed as a collection of facets. You can use `Path` or `ObjectIdentifier` to access objects. Objects can also have attributes, which are a user-defined unit of metadata. For example, the user object can have an attribute called *email-address*. Attributes are always associated with an object.

Policies

Policies are a specialized type of object that are useful for storing permissions or capabilities. Policies offer the `LookupPolicy` API action. The lookup policy action takes a reference to any object as its starting input. It then walks up the directory all the way to the root. The action collects any policy objects that it encounters on each path to the root. Cloud Directory does not interpret any of these policies in any way. Instead, Cloud Directory users interpret policies using their own specialized business logic.

For example, imagine a system that stores employee information. Employees are grouped together by job function. We want to establish different permissions for members of the Human Resources Group and the Accounting group. Members of the Human Resources group will have access to payroll information and the Accounting group will have access to ledger information. To establish these permissions, we attach policy objects to each of these groups. When it is time to evaluate a user's permissions, we can use the `LookupPolicy` API action on that user's object. The `LookupPolicy` API action walks the tree from the specified policy's object up to the root. It stops at each node and checks for any attached policies and returns those.

Policy Attachments

Policies can be attached to other objects in two ways: normal parent-child attachments and special policy attachments. Using normal parent-child attachments, a policy can be attached to a parent node. This is often useful to provide an easy mechanism to locate policies within your data directory. Policies cannot have children. Policies attached via parent-child attachments will not be returned during `LookupPolicy` API calls.

Policy objects can also be attached to other objects via policy attachments. You can manage these policy attachments using the `AttachPolicy` and `DetachPolicy` API actions. Policy attachments allow policy nodes to be located when you use the LookupPolicy API.

Policy Schema Specification

In order to start using policies, you must first add a facet to your schema that support creating policies. To accomplish this, create a facet setting the `objectType` of the facet to POLICY. Creating objects using a facet with the type POLICY ensures that the object has policy capabilities.

Policy facets inherit two attributes in addition to any attributes you add to the definition:

- **policy_type** (String, Required) – This is an identifier you can provide to distinguish between different policy uses. If your policies logically fall into clear categories, we encourage setting the policy type attribute appropriately. The `LookupPolicy` API returns the policy type of attached policies (see `PolicyAttachment`). This allows easy filtering of the specific policy type that you are looking for. It also allows you to use policy_type to decide how the document should be processed or interpreted.
- **policy_document** (Binary, Required) – You can store application specific data in this attribute, such as permission grants associated with the policy. You can also store application-related data in normal attributes on your facet, if you prefer.

Policy API Overview

A variety of specialized API actions are available for working with policies. For a list of available operations, see Amazon Cloud Directory Actions.

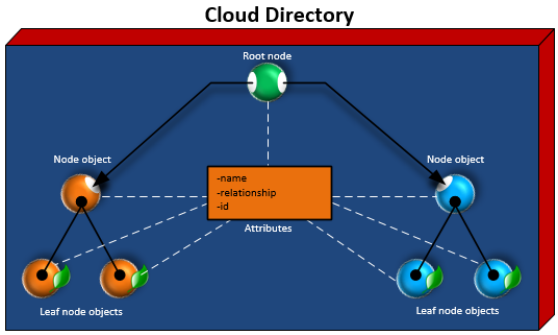
To create a policy object, use the `CreateObject` API action with an appropriate facet:

- To attach or detach a policy from an object, use the actions `AttachPolicy` and `DetachPolicy` respectively.
- To find policies that are attached to objects up the tree, use the `LookupPolicy` API action.
- To list the policies that are attached to a particular object, use the `ListObjectPolicies` API action.

For a list of operations and the permissions required to perform each API action, see Amazon Cloud Directory API Permissions: Actions, Resources, and Conditions Reference (p. 53).

Directory Structure

Data in a directory is structured hierarchically in a tree pattern consisting of nodes, leaf nodes, and links between the nodes, as shown in the following illustration. This is useful in application development to model, store, and quickly traverse hierarchical data.



Root Node

The root is the top node in a directory that is used to organize the parent and child nodes in the hierarchy. This is similar to how folders in a file system can contain subfolders and files.

Node

A node represents an object that can have child objects. For example, a node can logically represent a group of managers whereby various user objects are the children, or leaf nodes. A node object can only have one parent.

Leaf Node

A leaf node represents an object with no children that may or may not be directly connected to a parent node. For example, a user or device object. A leaf node object can have multiple parents. While leaf node objects are not required to be connected to a parent node, it is strongly recommended that you do so, since without a path from the root, the object can only be accessed by it's `NodeId`. If you misplace the id of such an Object, you will have no way to locate it again.

Node Link

The connection between one node and another. Cloud Directory supports a variety of link types between nodes, including parent-child links, policy links, and index attribute links.

Schemas

With Amazon Cloud Directory, schemas define what types of objects can be created within a directory (users, devices, and organizations), enforce validation of data for each object class, and handle changes to the schema over time. More specifically, a schema defines the following:

- One or more types of facets that may be mapped to objects within a directory (such as Person, Organization_Person)
- Attributes that may be mapped to objects within a directory (such as Name, Description). Attributes can be required or made optional on various types of facets, and are defined within the context of a facet.
- Constraints that may be enforced on object attributes (such as Required, Integer, String)

When a schema has been applied to a directory, all data within that directory must then conform to that applied schema. In this way, the schema definition is essentially a blueprint that can be used to construct multiple directories with applied schemas. Once built, those applied schemas may vary from the original blueprint, each in different ways.

Applied schemas can later be updated using versioning and then reapplied to all the directories that use it. For more information, see In-Place Schema Upgrade (p. 10).

Cloud Directory provides API operations to create, read, update, and delete schemas. This allows the contents of the schema to be easily consumed by programmatic agents. Such agents access the directory to discover the full set of facets, attributes, and constraints that apply to data within the directory. For more information about the schema APIs, see the Amazon Cloud Directory API Reference Guide.

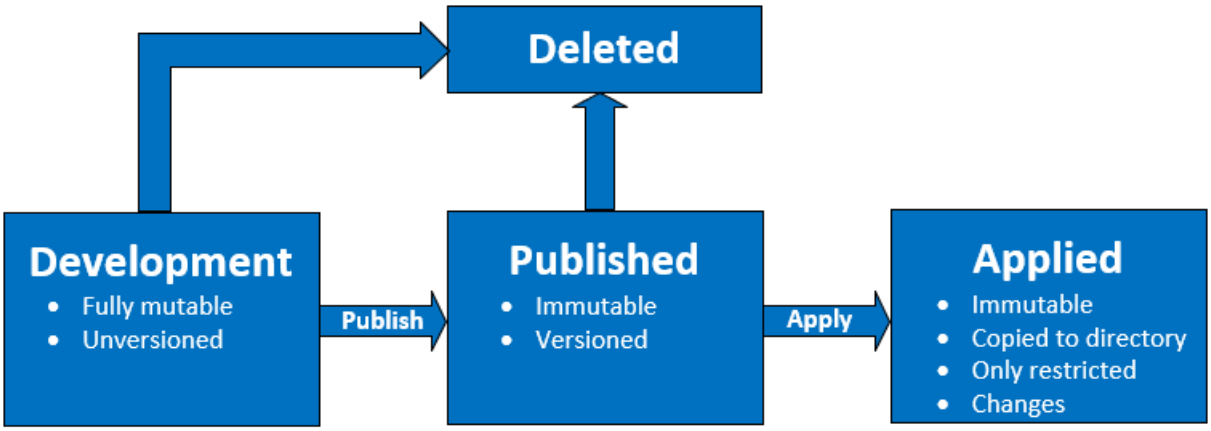
Cloud Directory supports uploading a compliant JSON file for schema creation. You can also create and manage schemas using the AWS Directory Services console. For more information, see Create an Amazon Cloud Directory (p. 3).

Topics

Schema Lifecycle

Cloud Directory offers a schema lifecycle to help with the development of schemas. This lifecycle consists of three states: Development, Published, and Applied. These states are designed to facilitate construction and distribution of schemas. Each of these states has different features aiding this effort.

The following diagram depicts possible transitions and verbiage. All schema transitions are copy-on-write. For example, publishing a development schema does not alter or remove the development schema.



You can delete a schema when it is in either the Development or Published state. Deleting a schema cannot be undone nor can it be restored once it has been deleted.

Schemas in Development, Published and Applied states have ARNs that represent them. These ARNs are used in API operations to describe the schema that the API operates on. It is easy to discern the state of a schema by looking at a schema ARN.

- Development: `arn:aws:clouddirectory:us-east-1:1234567890:schema/development/SchemaName`
- Published: `arn:aws:clouddirectory:us-east-1:1234567890:schema/published/SchemaName/Version`
- Applied: `arn:aws:clouddirectory:us-east-1:1234567890:directory/directoryid/schema/SchemaName/Version`

Development State

Schemas are initially created in the development state. Schemas in this state are fully mutable. You can freely add or remove facets and attributes. The majority of schema design occurs in this state. Schemas in this state have a name but no version.

Published State

The published schema state stores schemas that are ready to be applied to data directories. Schemas are published from the development state into the published state. You cannot change schemas in the published state. You can apply published schemas to any number of data directories.

Published and applied schemas must have a version associated with them. For more information about versions, see Schema Versioning (p. 10).

Applied State

A published schema can be applied to data directories. A schema that has been applied to a data directory is said to be applied. Once you apply a schema to a data directory, you can use the schema's facets when creating objects. You can apply multiple schemas to the same data directory. Only the following changes are permitted on an applied schema.

- Add a facet to an applied schema
- Add a non-required attribute to an applied schema

Facets

Facets are the most basic abstraction within a schema. They represent a set of attributes that can be associated with an object in the directory and are similar in concept to LDAP object classes. Each directory object may have up to a certain number of facets associated with it. For more information, see Amazon Cloud Directory Limits (p. 64).

Each facet maintains its own independent set of attributes. Each facet consists of fundamental metadata, such as the facet name, version information, and behaviors. The combination of schema ARNs, facets, and attributes define uniqueness on the object.

The set of object facets, their constraints, and the relationships between them constitute an abstract schema definition. Schema facets are used to define constraints over the following things:

1. Attributes allowed in an object
2. Policy types allowed to apply to an object

Once you have added the necessary facets to your schema, you can apply the schema to your directory and create the applicable objects. For example, you can define a device schema by adding facets such as computers, phones, and tablets. You can then use these facets to create computer objects, phone objects, and tablet objects in the directory to which the schema applies.

Cloud Directory's schema support makes it easy to add or modify facets and attributes without worrying about breaking applications. For more information, see In-Place Schema Upgrade (p. 10).

In-Place Schema Upgrade

Cloud Directory offers the updating of existing schema attributes and facets to help integrate your applications with AWS provided services. Schemas that are in either the published or applied states have versions and cannot be changed. For more information, see Schema Lifecycle (p. 8).

Schema Versioning

A schema version indicates a unique identifier for a schema that developers can specify when programming their applications to conform to certain rules and formatting of data. Two key differentiators in the way versioning works with Cloud Directory are important for developers to understand. These differentiators—major version and minor version—can determine how future schema upgrades impact your application.

Major Version

Major version is the version identifier used for tracking major version changes for a schema. It can be up to 10 characters in length. Different versions of the same schema are completely independent. For example, two schemas with the same name and different versions are treated as completely different schemas, which have their own namespaces.

Backward incompatible changes

We recommend making changes to the major version only when schemas are incompatible. For example, when changing the data type of an existing attribute (such as changing from `string` to `integer`) or dropping a mandatory attribute from your schema. Backward-incompatible changes require directory data migration from a previous schema version to the new schema version.

Minor Version

Minor version is the version identifier used for in-place upgrading of schemas or when you want to make backward-compatible upgrades such as adding additional attributes or adding facets. An upgraded schema using a minor version can be applied in place across all directories that use it without breaking any running applications. This includes directories that are used in production environments. For an example use case, see "How to Easily Apply Amazon Cloud Directory Schema Changes with In-Place Schema Upgrades" in the Cloud Directory Blog.

The minor version information and history is saved along with the other schema information in the schema metadata repository. No minor version information is retained in the objects. The advantage of introducing minor version is that client code works seamlessly as long as the major version is not changed.

Minor Version Limits

Cloud Directory retains and therefore limits up to five minor versions. However, minor version limits are enforced differently for published and applied schemas in the following ways:

- **Applied schemas:** Once the minor version limit has been exceeded, Cloud Directory deletes the oldest minor version automatically.
- **Published schemas:** Once the minor version limit has been exceeded, Cloud Directory does not delete any of the minor versions but it does inform the user via a `LimitExceededException` that the limit has been exceeded. Once you exceed the minor version limits, you can either delete the schema by using the DeleteSchema API or request a limit raise.

Using the Schema Upgrade API Operations

You can use the `UpgradePublishedSchema` API call to upgrade published schemas. Schema upgrades are applied in place to the directories that rely on it using the `UpgradeAppliedSchema` API call. You can also fetch the major and minor version of an applied schema by calling `GetAppliedSchemaVersion`s. Or view the associated schema ARNs and schema revision history for a directory by calling `ListAppliedSchemaArns`. Cloud Directory maintains the five most recent versions of applied schema changes.

For an illustrative example, see "How to Easily Apply Amazon Cloud Directory Schema Changes with In-Place Schema Upgrades" in the Cloud Directory Blog. The blog post will demonstrate how you perform an in-place schema upgrade and use schema versions in Cloud Directory. It covers how to add additional attributes to an existing facet, add a new facet to a schema, publish the new schema, and apply it to running directories to complete the upgrading of a schema in-place. It also shows how to view the version history of a directory schema, which helps to ensure the directory fleet is running the same version of the schema and has the correct history of schema changes applied to it.

Managed Schema

Cloud Directory makes it easy for you to rapidly develop applications by using a managed schema. With a managed schema, you can create a directory and start creating and retrieving objects from it at a faster pace. For more information, see Create Your Directory (p. 43).

Currently, there is one managed schema, called the `QuickStartSchema`. You can build a rich hierarchical data model and establish relationships across objects by using constructs such as Typed Links (p. 28). You can then query for any information in your data by traversing the hierarchy.

The `QuickStartSchema` managed schema is represented by the following JSON:

```
QuickStartSchema: {
    "facets": {
        "DynamicObjectFacet": {
            "facetStyle": "DYNAMIC"
        },
        "DynamicTypedLinkFacet": {
            "facetAttributes": {
                "DynamicTypedLinkAttribute": {
                    "attributeDefinition": {
                        "attributeRules": {},
                        "attributeType": "VARIANT",
                        "isImmutable": false
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "identityAttributeOrder": [
                "DynamicAttribute"
            ]
        }
    }
}
```

QuickStartSchema ARN

The `QuickStartSchema` managed schema uses the following ARN:

```
String QUICK_START_SCHEMA_ARN = "arn:aws:clouddirectory:::schema/managed/
quick_start/1.0/001" ;
```

For example, you could use this ARN to create a directory called `ExampleDirectory` as shown below:

```
CreateDirectoryRequest createDirectoryRequest = new CreateDirectoryRequest()
    .withName("ExampleDirectory") // Directory name
    .withSchemaArn(QUICK_START_SCHEMA_ARN);
```

Facet Styles

There are two different styles that you can define on any given facet, `Static` and `Dynamic`.

Static Facets

Static facets are the best choice when you have all the details of your data model for your directory, such as a list of attributes with their data types, and you also want to define constraints for your attributes such as mandatory or unique fields. Cloud Directory will enforce the data constraints and rule checking during your object creation or change.

Dynamic Facets

You can use a dynamic facet when you need flexibility to change the number of attributes or change the data values being stored within your attributes. Cloud Directory does not enforce any data constraints and rule checking during your object creation or change.

After creating a schema with dynamic facets, you can define any attributes that you need while creating objects. Cloud Directory will accept the attributes as key-value pairs and store them on their provided objects.

You can add a dynamic facet to a new or existing schema. You can also combine the static and dynamic facets within a single schema to get benefits for each style of facet within your directory.

When you create any attribute using Dynamic facet, they are created as `Variant` data type. To store values for the attribute defined as a `Variant` data type, you can use values of any of the primitive data types supported in Cloud Directory, such as `String` or `Binary`. Over time, you can also change the value of the attribute to another datatype. There is no enforcement of data validation.

You can use dynamic facets to define objects of the following type:

- `NODE`
- `LEAF_NODE`
- `POLICY`

For additional details about managed schemas, dynamic facets or variant data types and to see example use cases, see How to rapidly develop applications on Amazon Cloud Directory using AWS Managed Schema in the Amazon Cloud Directory blog.

Sample Schemas

Cloud Directory comes ready with sample schemas for Organizations, Persons, and Devices. The following section lists the various sample schemas and lists the differences for each.

Organizations

The following tables list the facets that are included in the *Organizations* sample schema.

"Organization" Facet	Data Type	Length	Require Behavi	Description
account_id	String	1024	N	Unique id for Organization
account_name	String	1024	N	Name of Organization
organization_status	String	1024	N	Status such as 'active', 'suspended', 'inactive', 'closed'
mailing_address (street1)	String	1024	N	A physical mailing address for this company/entity
mailing_address (street2)	String	1024	N	A physical mailing address for this company/entity
mailing_address (city)	String	1024	N	A physical mailing address for this company/entity
mailing_address (state)	String	1024	N	A physical mailing address for this company/entity
mailing_address (country)	String	1024	N	A physical mailing address for this company/entity
mailing_address (postal_code)	String	1024	N	A physical mailing address for this company/entity
email	String	1024	N	Email id for Organization
web_site	String	1024	N	Website URL

"Organization" Facet	Data Type	Length	Require Behavi	Description
telephone_number	String	1024	N	Telephone number for Organization
description	String	1024	N	Description for Organization

"Legal_Entity" Facet	Data Type	Length	Require Behavi	Description
registered_company_name	String	1024	N	Legal entity name
mailing_address (street1)	String	1024	N	A physical registered address for this company/entity
mailing_address (street2)	String	1024	N	A physical registered address for this company/entity
mailing_address (city)	String	1024	N	A physical registered address for this company/entity
mailing_address (state)	String	1024	N	A physical registered address for this company/entity
mailing_address (country)	String	1024	N	A physical registered address for this company/entity
mailing_address (postal_code)	String	1024	N	A physical registered address for this company/entity
industry_vertical	String	1024	N	Industry Segment
billing_currency	String	1024	N	Billing currency
tax_id	String	1024	N	Tax identification number

Person

The following tables list the facets that are included in the *Person* sample schema.

"Person" Facet	Data Type	Length	Required Behavior?	Description
display_name	String	1024	N	The name of the user, suitable for display to end-users.
first_name	String	1024	N	The given name of the User, or first name in most western languages
last_name	String	1024	N	The family name of the User, or last name in most western languages
middle_name	String	1024	N	The middle name(s) of the User

"Person" Facet	Data Type	Length	Required Behavior?	Description
nickname	String	1024	N	The casual way to address the user in real life, such as, "Bob" or "Bobby" instead of "Robert"
email	String	1024	N	Email address for the user
mobile_phone_number	String	1024	N	Phone number for the user
home_phone_number	String	1024	N	Phone number for the user
username	String	1024	Y	unique identifier for the user
profile	String	1024	N	A URI that is a uniform resource locator and that points to a location representing the user's online profile (such as a webpage)
picture	String	1024	N	A URI that is a uniform resource locator that points to a resource location representing the user's image.
website	String	1024	N	URL
timezone	String	1024	N	The User's time zone
locale	String	1024	N	Used to indicate the User's default location for purposes of localizing such items as currency, date time format, or numerical representations.
address (street1)	String	1024	N	A physical mailing address for this user.
address (street2)	String	1024	N	A physical mailing address for this user.
address (city)	String	1024	N	A physical mailing address for this user.
address (state)	String	1024	N	A physical mailing address for this user.
address (country)	String	1024	N	A physical mailing address for this user.
address (postal_code)	String	1024	N	A physical mailing address for this user.
user_status	String	1024	N	Value indicating the user's administrative status

"Organization_Person" Facet	Data Type	Length	Required Behavior?	Description
title	String	1024	N	Title in organization
preferred_language	String	1024	N	Indicates the user's preferred written or spoken languages and is generally used for selecting a localized user interface.
employee_id	String	1024	N	A string identifier, typically numeric or alphanumeric, assigned to a person
cost_center	Integer	1024	N	Identifies the cost center
department	String	1024	N	Identifies the name of a department
manager	String	1024	N	The user's manager
company_name	String	1024	N	Identifies the name of an organization
company_address (street1)	String	1024	N	A physical mailing address for the organization
company_address (street2)	String	1024	N	A physical mailing address for the organization
company_address (city)	String	1024	N	A physical mailing address for the organization
company_address (state)	String	1024	N	A physical mailing address for the organization
company_address (country)	String	1024	N	A physical mailing address for the organization
company_address (postalCode)	String	1024	N	A physical mailing address for the organization

Device

The following table lists the facet that is included in the *Device* sample schema.

"Device" Facet	Data Type	Length	Required Behavior?	Description
device_id	String	1024	N	Alpha-numeric unique device id
name	String	1024	N	Friendly name for device
description	String	1024	N	Description for device
X.509_certificates	String	1024	N	X.509 Certificate

"Device" Facet	Data Type	Length	Required Behavior?	Description
device_version	String	1024	N	Device version
device_os_type	String	1024	N	Operating System on device
device_os_version	String	1024	N	Operating System version number on device
serial_number	String	1024	N	Serial number of device
device_status	String	1024	N	Status for device (such as active, not_active, suspended, shutdown, off)

Custom Schemas

The first step in creating a custom schema is to define exactly what fields you must index. These required fields form your schema's skeleton elements, to which you add your own fields. Map the name and type of each field (such as string, integer, Boolean) to your object's structure. You can define a schema with types and constraints and then apply them to a directory. Once defined, Cloud Directory performs validation for attributes.

For more information, see Create a Schema (p. 2).

Attribute References

Amazon Cloud Directory facets contain attributes. Attributes can be either an attribute definition or an attribute reference. Attribute definitions are attributes that declare their name and primitive type (string, binary, Boolean, DateTime, or number). They can also optionally declare their required behavior, default value, immutable flag, and attribute rules (such as min/max length).

Attribute references are attributes that derive their primitive type, default value, immutable flag and attribute rules from another preexisting attribute definition. Attribute references do not have their own primitive type, default values, immutable flag or rules, since those properties come from the target attribute definition.

Attribute references may override the required behavior of a target definition (more details about this below).

When you create an attribute reference, you provide only an attribute name and the target attribute definition (which includes the facet name and attribute name of the target attribute definition). Attribute references may not refer to other attribute references. Also, at this time, attribute references may not target attribute definitions from a different schema.

You can use an attribute reference when you want two or more attributes on an object to refer to the same storage location. For example, imagine an object that has a User facet and an EnterpriseUser facet applied. The User facet has a FirstName attribute definition, while the EnterpriseUser facet has an attribute reference pointing at User.FirstName. Since both FirstName attributes refer to the same storage location on the object, any change to either the User.FirstName or the EnterpriseUser.FirstName has the same effect.

API Example

The following example demonstrates the use of attribute references using the Cloud Directory API. In this example, a base facet contains an attribute definition, and another facet contains an attribute referring to an attribute in the base facet. Note that the reference attribute can be marked Required while the base facet is Not Required.

```
    // create base facet
    CreateFacetRequest req1 = new CreateFacetRequest()
      .withSchemaArn(devSchemaArn)
      .withName("baseFacet")
      .withAttributes(List(
        new FacetAttribute()
          .withName("baseAttr")
          .withRequiredBehavior(RequiredAttributeBehavior.NOT_REQUIRED)
          .withAttributeDefinition(new
FacetAttributeDefinition().withType(FacetAttributeType.STRING))))
      .withObjectType(ObjectType.DIRECTORY)
    cloudDirectoryClient.createFacet(req1)

    // create another facet that refers to the base facet
    CreateFacetRequest req2 = new CreateFacetRequest()
      .withSchemaArn(devSchemaArn)
      .withName("facetA")
      .withAttributes(List(
        new FacetAttribute()
          .withName("ref")
          .withRequiredBehavior(RequiredAttributeBehavior.REQUIRED_ALWAYS)
          .withAttributeReference(new FacetAttributeReference()
            .withTargetFacetName("baseFacet")
            .withTargetAttributeName("baseAttr"))))
      .withObjectType(ObjectType.DIRECTORY)
    cloudDirectoryClient.createFacet(req2)
```

JSON Example:

The following example demonstrates the use of attribute references in a JSON model. The schema represented by this model is identical to the model above.

```
{
 "facets" : {
   "baseFacet" : {
     "facetAttributes" : {
       "baseAttr" : {
         "attributeDefinition" : {
           "attributeType" : "STRING"
         },
         "requiredBehavior" : "NOT_REQUIRED"
       }
     },
     "objectType" : "DIRECTORY"
   },
   "facetA" : {
     "facetAttributes" : {
       "ref" : {
         "attributeReference" : {
           "targetFacetName" : "baseFacet",
           "targetAttributeName" : "baseAttr"
         },
         "requiredBehavior" : "REQUIRED_ALWAYS"
       }
```

```
      },
      " objectType" : "DIRECTORY"
    }
}
```

Attribute reference considerations

Attribute references must target a preexisting attribute definition in the same schema.

- Attribute references may target a preexisting attribute definition in the same facet or a different facet.
- Attribute references may not target other attribute references.
- Facets containing attribute definitions that are the target of another facet's attribute reference cannot be deleted until all references have been deleted.

You can use attribute references the same way that you use traditional attribute definitions, by creating objects or applying facets to existing objects.

> **Note**
> You can apply facets with references to other facets but are not required to apply the target facets directly. When the target facet is not applied, there is no change to the behavior of the attribute reference. (You need to apply target facets only when you want the other attributes on that facet to exist on the object.)

Setting Attribute Reference Values

You can call the UpdateObjectAttributes API action when you want to change the value of an attribute. Updating (or deleting) either the definition or any other reference to that same definition on that object has the same effect.

Getting Attribute Reference Values

You can call the ListObjectAttributes API action to retrieve storage aliases. This call returns a list of tuples, each of which contains an attribute key and its associated value. The attribute keys correspond to the list of storage aliases present on that object.

> **Note**
> It is possible for an attribute key to be returned for a facet that was not explicitly applied to an object. This can happen when attribute references target facets that are not applied to the object.

For example, imagine that you have a User facet and an EnterpriseUser facet. The EnterpriseUser.FirstName attribute refers to User.FirstName. You then apply both the User and EnterpriseUser facets to an object, set User.FirstName to Robert, and later set EnterpriseUser.FirstName to Bob. When you call ListObjectAttributes you see only "User.FirstName = Bob" since there is only one storage alias for both FirstName attributes.

Using Indexes with Attribute References

You can can create indexes with an attribute definition only, not a reference. Listing an index does not return attribute keys for attribute references. But it does return attribute keys for any attribute definitions that are targeted by references existing on the indexed object. In other words, at the index layer, attribute references are treated merely as an alternative identifier for an attribute, which gets resolved to the correct attribute definition identifier at runtime.

For example, imagine you have an Index for facet User attribute FirstName. You attach an object with only the EnterpriseUser facet applied. You then set the value for that object's EnterpriseUser.FirstName attribute to Bob. Finally, you call ListIndex action. The results contain only "User.FirstName = Bob".

Required Behavior for Attribute References

An attribute references can have a required behavior that is different from its target attribute definition. This allows a base definition to be optional, while a reference to that same definition can be required. When an object has a base definition and one or more references to the same base definition, the base definition and all references must adhere to the strongest required behavior present across all related attributes.

- As with attribute definitions, you must provide values for any required attribute definitions when you create the object or when you add a facet to an existing object.
- As a convenience, when more than one attribute on an object refers to the same storage location, you only need to provide a value for one of the attributes for that storage location.
- Similarly, if you do provide multiple values for the same storage location, the values must be equal.

Attribute Rules

Rules describe permissible values of an attribute type and constrain the values that are allowed for any particular attribute. You must specify rules as part of an attribute definition when you create a facet. Cloud Directory supports the following rule types:

- String length
- Binary length
- String from set
- Number comparison

String length

Constrains the length of a string attribute value.

Allowed rule parameter keys: min, max

Allowed rule parameter values: number

Binary length

Constrains the byte array length of a binary attribute value.

Allowed rule parameter keys: min, max

Allowed rule parameter values: number

String from set

Constrains the value of a string attribute to the allowed set of specified strings.

Allowed rule parameter keys: allowedValues

Allowed rule parameter values: Set of strings with each string to be UTF-8 encoded

Allowed values are comma delimited and can be wrapped in quotes. This is useful when allowed values include comma's. For example:

- One,two,three = matches One two or three
- "with,comma","withoutcomma" = matches "with,comma" or "withoutcomma"

- with”quote,withoutquote matches ‘with”quote’ or ‘withoutquote’

Number comparison

Constraints the numeric value allowed for a number attribute.

Allowed rule parameter keys: min, max

Allowed rule parameter values: number

Format Specification

A Cloud Directory schema adds structure to the data in your data directories. Cloud Directory provides two mechanisms for you to define your schema. Developers can use specific API operations to construct a schema or they can upload a schema entirely using schema upload capabilities. Schema documents can be uploaded via API calls or through the console. This section describes the format to use when you upload entire schema documents.

JSON Schema Format

A schema document is a JSON document in the following overall format.

```
{
    "facets": {
        "facet name": {
            "facetAttributes": {
                "attribute name": Attribute JSON Subsection
            }
        }
    }
}
```

A schema document contains a map of facet names to facets. Each facet in turn contains a map containing attributes. All facet names within a schema must be unique. All attribute names within a facet must be unique.

Attribute JSON Subsection

Facets contain attributes. Each attribute defines the type of value that can be stored on an attribute. The following JSON format describes an attribute.

```
{
    "attributeDefinition": Attribute Definition Subsection,
    "attributeReference": Attribute Reference Subsection,
    "requiredBehavior": "REQUIRED_ALWAYS" or "NOT_REQUIRED"
}
```

You must provide either an attribute definition or an attribute reference. See related subsections for more information about each.

The required behavior field indicates whether this attribute is required or not. You must provide this field. Possible values are as follows:

- `REQUIRED_ALWAYS`: This attribute must be provided when the object is created or a facet is added to the object. You cannot remove this attribute.

- `NOT_REQUIRED`: This attribute may or may not be present.

Attribute Definition Subsection

An attribute defines the type and the rules associated with an attribute value. The following JSON layout describes the format.

```
{
    "attributeType": One of "STRING", "NUMBER", "BINARY", "BOOLEAN" or "DATETIME",
    "defaultValue": Default Value Subsection,
    "isImmutable": true or false,
    "attributeRules": "Attribute Rules Subsection"
}
```

Default Value Subsection

Specify exactly one of the following default values. Long values and Boolean values should be provided outside of quotes (as their respective Javascript types instead of strings). Binary values are provided using a URL-safe Base64 encoded string (as described in RFC 4648). Datetimes are provided in the number of milliseconds since the epoch (00:00:00 UTC on Jan 1, 1970).

```
{
 "stringValue": "a string value",
 "longValue": an integer value,
 "booleanValue": true or false,
 "binaryValue": a URL-safe Base64 encoded string,
 "datetimeValue": an integer value representing milliseconds since epoch
}
```

Attribute Rules Subsection

Attributes rules define constraints on attribute values. You can define multiple rules for each attribute. Attribute rules contain a rule type and a set of parameters for the rule. You can find more details in the Attribute Rules (p. 20) section.

```
{
    "rule name": {
        "parameters": {
            "rule parameter key 1": "value",
            "rule parameter key 2": "value"
        },
        "ruleType": "rule type value"
    }
}
```

Attribute Reference Subsection

Attribute references are an advanced feature. They allow multiple facets to share an attribute definition and stored value. See the Attribute References (p. 17) section for more information. You can define an attribute reference in JSON schema with the following template.

```
{
 "targetSchemaArn": "schema ARN"
 "targetFacetName": "facet name"
 "targetAttributeName": "attribute name"
}
```

Schema Document Examples

The following are samples of schema documents that show valid JSON formatting.

Note
All values expressed in the `allowedValues` string must be comma separated and be without spaces. For example, `"SENSITIVE,CONFIDENTIAL,PUBLIC"`.

Basic Schema Document

```
{
    "facets": {
        "Employee": {
            "facetAttributes": {
                "Name": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": false,
                        "attributeRules": {
                            "NameLengthRule": {
                                "parameters": {
                                    "min": "3",
                                    "max": "100"
                                },
                                "ruleType": "STRING_LENGTH"
                            }
                        }
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "EmailAddress": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {
                            "EmailAddressLengthRule": {
                                "parameters": {
                                    "min": "3",
                                    "max": "100"
                                },
                                "ruleType": "STRING_LENGTH"
                            }
                        }
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "Status": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": false,
                        "attributeRules": {
                            "rule1": {
                                "parameters": {
                                    "allowedValues": "ACTIVE,INACTIVE,TERMINATED"
                                },
                                "ruleType": "STRING_FROM_SET"
                            }
                        }
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "objectType": "LEAF_NODE"
```

```
        },
        "DataAccessPolicy": {
            "facetAttributes": {
                "AccessLevel": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {
                            "rule1": {
                                "parameters": {
                                    "allowedValues": "SENSITIVE,CONFIDENTIAL,PUBLIC"
                                },
                                "ruleType": "STRING_FROM_SET"
                            }
                        }
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "objectType": "POLICY"
        },
        "Group": {
            "facetAttributes": {
                "Name": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "objectType": "NODE"
        }
    }
}
```

Schema Document with Typed Links

```
{
    "sourceSchemaArn": "",
    "facets": {
        "employee_facet": {
            "facetAttributes": {
                "employee_login": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "employee_id": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "employee_name": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {}
```

```
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            },
            "employee_role": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
                    "attributeRules": {}
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            }
        },
        "objectType": "LEAF_NODE"
    },
    "device_facet": {
        "facetAttributes": {
            "device_id": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
                    "attributeRules": {}
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            },
            "device_type": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
                    "attributeRules": {}
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            }
        },
        "objectType": "NODE"
    },
    "region_facet": {
        "facetAttributes": {},
        "objectType": "NODE"
    },
    "group_facet": {
        "facetAttributes": {
            "group_type": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
                    "attributeRules": {}
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            }
        },
        "objectType": "NODE"
    },
    "office_facet": {
        "facetAttributes": {
            "office_id": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
                    "attributeRules": {}
                },
                "requiredBehavior": "REQUIRED_ALWAYS"
            },
            "office_type": {
                "attributeDefinition": {
                    "attributeType": "STRING",
                    "isImmutable": true,
```

```
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "office_location": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": true,
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "objectType": "NODE"
        }
    },
    "typedLinkFacets": {
        "device_association": {
            "facetAttributes": {
                "device_type": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": false,
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                },
                "device_label": {
                    "attributeDefinition": {
                        "attributeType": "STRING",
                        "isImmutable": false,
                        "attributeRules": {}
                    },
                    "requiredBehavior": "REQUIRED_ALWAYS"
                }
            },
            "identityAttributeOrder": [
                "device_label",
                "device_type"
            ]
        }
    }
}
```

Directory Objects

Developers model directory objects using extensible schemas to enforce data correctness constraints automatically, making it easier to program for. Amazon Cloud Directory offers rich information lookup based on your defined indexed attributes, thus enabling fast tree traversals and searches within the directory trees. Cloud Directory data is encrypted at rest and in transit.

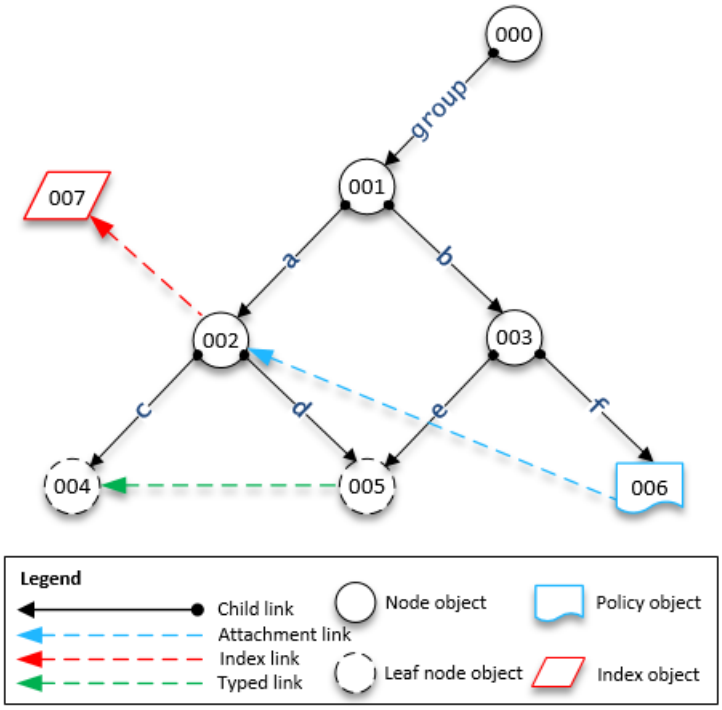
An object is a basic element of Cloud Directory. Each object has a globally unique identifier, which is specified by the object Identifier. An object is a collection of zero or more facets with their attribute keys and values. An object can be created from one or more facets within a single applied schema or from facets of multiple applied schemas. During object creation, you must specify all required attribute values. Objects can have a limited number of facets. For more information, see Amazon Cloud Directory Limits (p. 64).

An object can be a regular object, a policy object, or an index object. An object can also be a node object or a leaf node object. The type of the object is inferred from the object type of the facets attached to it.

Topics

Links

A link is a directed edge between two objects that define a relationship. Cloud Directory currently supports the following link types.



Child Links

A child link creates a parent–child relationship between the objects it connects. For example, in the above illustration child link **b** connects objects `001` and `003`. Child links define the hierarchy in Cloud Directory. Child links have names when they participate in defining the path of the object that the link points to.

Attachment Links

An attachment link applies a leaf node policy object to another leaf node or a node object. Attachment links do not define the hierarchical structure of Cloud Directory. For example, in the above illustration, attachment link applies the policy stored in policy leaf node object `006` on node object `002` . Each object can have multiple policies attached but not more than one policy of any given policy type can be attached.

Index Links

Index links provide rich information lookup based on an index object and your defined indexed attributes, thus enabling fast tree traversals and searches within the directory trees. Conceptually, indexes are similar to nodes with children: The links to the indexed nodes are labeled according to the indexed attributes, rather than being given a label when the child is attached. However, index links are not parent-child edges, and have their own set of enumeration API operations. For more information, see Indexing and Search (p. 40).

Typed Links

Typed Links enable you to establish a relationship between objects within or across hierarchies in Cloud Directory. You can then use these relationships to query for information, such as *Which users have device 'xyz'* or *What devices are owned by user 'abc'*.

You can use typed links to model relationships between different objects in your directory. For example, in the illustration above, consider the relationship between object `004`, which represents a user, and object `005`, which represents a device.

We might use a typed link to model an ownership relationship between the two objects. We could add attributes to the typed link to represent the cost of a purchase, whether the device is rented or purchased. There are two types of attributes associated with typed links:

- **Identity-based attributes –** An attribute of a typed link that distinguishes it from other links (For example, Child, Attachment, Index links). Each typed link facet defines an ordered set of identity attributes. The identity of a typed link is the source object id, a facet identifier (type), the values of its identity attributes (defined by its facet), and the target object id. Identifiers must be unique within a single directory.
- **Optional attributes -** An attribute that stores tracking characteristics about the typed link that are unrelated to the identity of the link. For example, an optional attribute might identify the date the typed link was first established or when it was last modified.

As with objects, you must create a typed link facet using the `CreateTypedLinkFacet` API to define the typed link structure and its attributes. Typed link facets require a unique facet name and set of attributes that are associated with the link. When designing your typed link structure, you can define an ordered set of attributes on the typed link facet. To view a typed links sample schema, see Schema Document with Typed Links (p. 24).

Typed link attributes can be used when you need to do any of the following:

- Allow for filtering of incoming or outgoing typed links. For more information, see Typed Link Listing (p. 30).
- Represent the relationship between two objects.
- Track administrative data about your typed link, such as the date the link was created.

Consider the following when deciding if typed links are right for your use case:

- Typed links cannot be used in path-based object specification. Instead, you must select typed links using the `ListOutgoingTypedLinks` or `ListIncomingTypedLinks` API operations.
- Typed links do not participate in `LookupPolicy` or `ListObjectParentPaths` API operations.
- Typed links between the same two objects and in the same direction may not have the same attribute values. This can help avoid duplicated typed links between the same objects.
- Additional attributes can be used when you want to add optional information.
- The combined size of all identity attribute values is limited to 64 bytes. For more information, see Amazon Cloud Directory Limits (p. 64).

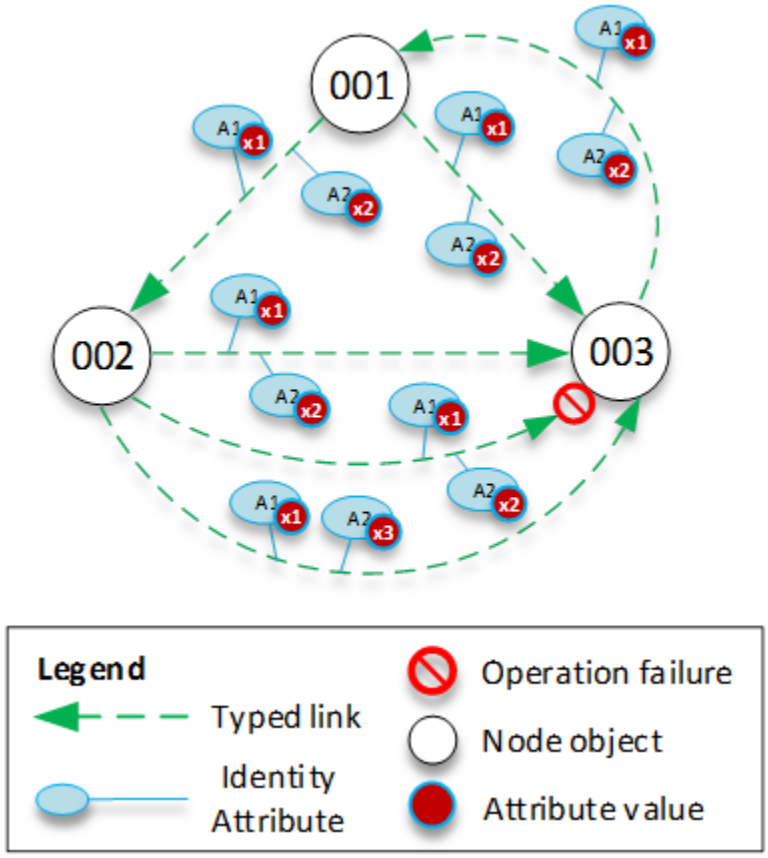
Related Cloud Directory Blog Article

- Use Amazon Cloud Directory Typed Links to Create and Search Relationships Across Hierarchies

Typed Link Identity

Identity is what uniquely defines whether a typed link can exist between two objects. The exception is when you connect two objects in one direction with the exact same attribute values. Attributes must be configured as `REQUIRED_ALWAYS`.

Typed links that are created from different typed link facets never conflict with each other. For example, consider the following diagram:



- Object `001` has typed links and attributes (A1 and A2) with the same attribute values (x1 and x2) going to different objects (`002` and `003`). This operation would succeed.
- Objects `002` and `003` have a typed link between them. This operation would fail because two typed links in the same direction with the same attributes cannot exist between objects.
- Objects `001` and `003` have two typed links between them with the same attributes. However, since the links go in different directions, this operation would succeed.
- Objects `002` and `003` have typed links between them with the same value for A1 but different values for A2. Typed link identity considers all attributes so this operation would succeed.

Typed Link Rules

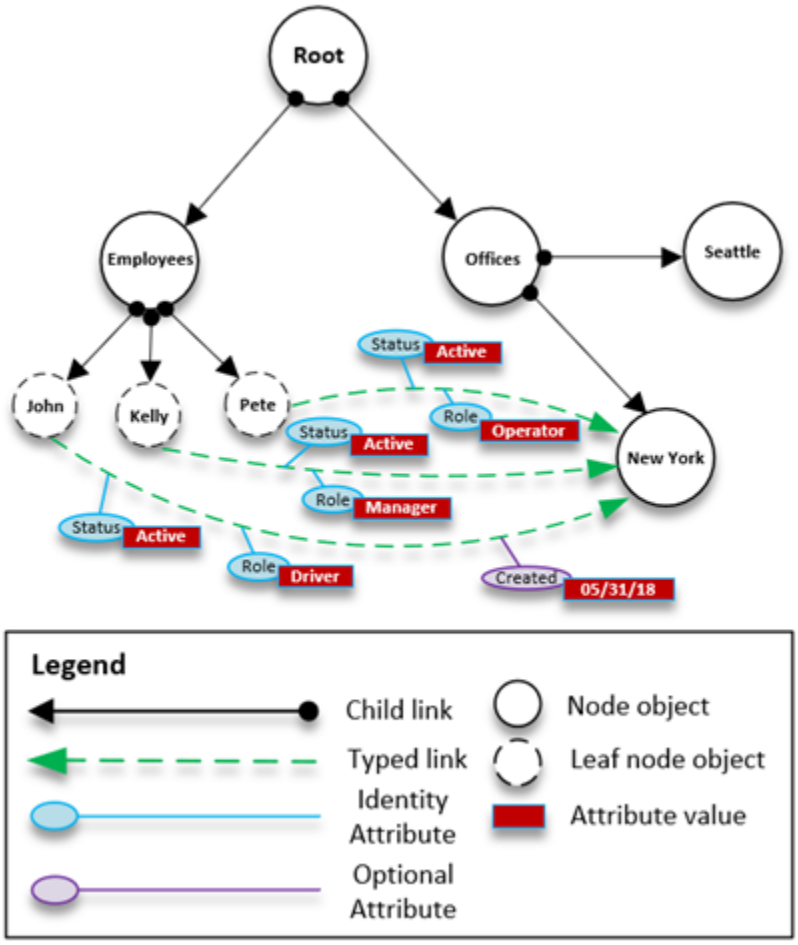
You can add rules to typed link attributes when you want to add restrictions to link attributes. These rules are equivalent to rules on object attributes. For more information, see Attribute Rules (p. 20).

Typed Link Listing

Cloud Directory provides API operations that you can use to select incoming or outgoing typed links from an object. You can select a specific subset of typed links rather than iterating over every typed link. You can also specify a particular typed link facet to filter only typed links of that type.

You can filter typed links based on the order that the attributes are defined on the typed link facet. You can provide range filters for multiple attributes. When providing ranges to a typed link selection, any inexact ranges much be specified at the end. Any attributes with no range specified are presumed to match the entire range. Filters are interpreted in the order of the attributes that are defined on the typed link facet, not the order they are supplied to any API calls.

For example, in the following diagram, consider a Cloud Directory that is used to store information about Employees and their Abilities.



Let's say we model our employee's capabilities with a typed link named `EmployeeCapability`, which is configured with three string attributes: `Status`, `Role` and `Created`. The following filters are supported on `ListIncomingTypedLinks` and `ListOutgoingTypedLinks` API operations.

- Facet = `EmployeeCapability`, Status = `Active`, Role = `Driver`
  - Selects active employees who are drivers. This filter includes two exact matches.
- Facet = `EmployeeCapability`, Status = `Active`, Role = `Driver`, Created = `05/31/18`
  - Selects active employees who are drivers and who's facets were created on or after May 31st, 2018.
- Facet = `EmployeeCapability`, Status = `Active`
  - Selects all active employees.
- Facet = `EmployeeCapability`, Status = `Active`, Role = `A` to `M`
  - Selects active employees with roles starting with `A` through `M`.
- Facet = `EmployeeCapability`
  - This selects all typed links of the `EmployeeCapability` type.

The following filters would **NOT** be supported:

- Facet = `EmployeeCapability`, Status between `A` to `C`, Role = `Driver`
  - This filter is not allowed because any ranges must appear at the end of the filter.

- Facet = `EmployeeCapability`, Role = `Driver`
  - This filter is not allowed because the implicit status range is not an exact match and does not appear at the end of the list of ranges.
- Status = `Active`
  - This filter is not allowed because the typed link facet is not specified.

Typed Link Schema

You can create typed link facets in two ways. You can manage your typed link facets from individual API calls, including `CreateTypedLinkFacet`, `DeleteTypedLinkFacet`, and `UpdateTypedLinkFacet`. You can also upload a JSON document that represents your schema in a single `PutSchemaFromJson` API call. For more information, see JSON Schema Format (p. 21). To view a typed links sample schema, see Schema Document with Typed Links (p. 24).

The types of changes allowed at different phases of the schema development lifecycle are similar to changes that are allowed for object facet manipulation. Schemas in the development state support any changes. Schemas in the published state are immutable and no changes are supported. Only certain changes are allowed to schemas that are applied to a data directory. Once you set the order and attributes on an applied typed link facet, that order cannot be changed.

Two other API operations list facets and their attributes:

- `ListTypedLinkFacetAttributes`
- `ListTypedLinkFacetNames`

Typed Link Interaction

Once a typed link facet has been created, you are ready to start creating and interacting with typed links. To attach and detach typed links, use the `AttachTypedLink` and `DetachTypedLink` API operations.

The `TypedLinkSpecifier` is a structure that contains all the information to uniquely identify a typed link. Within that structure you can find `TypedLinkFacet`, `SourceObjectID`, `DestinationObjectID`, and `IdentityAttributeValues`. These are used to uniquely specify the typed link being operated on. The `AttachTypedLink` API operation returns a typed link specifier while the `DetachTypedLink` API operation accepts one as input. Similarly, the `ListIncomingTypedLinks` and `ListOutgoingTypedLinks` API operations provide typed link specifiers as output. You can construct a typed link specifier from scratch as well. The full list of typed link-related API operations, include the following:

- `AttachTypedLink`
- `CreateTypedLinkFacet`
- `DeleteTypedLinkFacet`
- `DetachTypedLink`
- `GetLinkAttributes`
- `GetTypedLinkFacetInformation`
- `ListIncomingTypedLinks`
- `ListOutgoingTypedLinks`
- `ListTypedLinkFacetNames`
- `ListTypedLinkFacetAttributes`
- `UpdateLinkAttributes`
- `UpdateTypedLinkFacet`

Note
Attribute references and updating typed links are not supported. To update a typed link, you must remove it and add the updated version.

Range Filters

Several Cloud Directory list APIs allow specifying a filter in the form of a range. These filters allow you to efficiently select subsets of the links attached to the specified node.

Ranges are generally supplied as a map (array of key-value pairs) whose keys are attribute identifiers and whose values are the corresponding ranges. This allows filtering links whose identities consist of one or more attributes. For example, a TypedLink set up to model a Role relationship for determining permissions might have both RoleType and Authorizer attributes. A `ListOutgoingTypedLinks` call could then specify ranges to filter the result to RoleType:"Admin" and Authorizer:"Julia". The map of ranges used to filter a single list request must contain only attributes that define the link's identity (an index's OrderedIndexedAttributeList or a TypedLink's IdentityAttributeOrder), but it need not contain ranges for all of them. Missing ranges will automatically be filled in with ranges that span all possible values (from FIRST to LAST).

If you think of each attribute as defining an independent flat domain of values, the range structures define two logical points in that domain — the start and end points — and the range matches all of the possible points in between those points. The StartValue and EndValue of the range structure define the basis for these two points with the "modes" further refining them to indicating whether each point itself is to be included or excluded from the range. In the RoleType:"Admin" example above, the values for the RoleType attribute would both be "Admin", and the modes are both "INCLUSIVE" (written as ["Admin" to "Admin"]). A filter for a ListIndex call where the index is defined on the LastName of a User facet might use StartValue="D", StartMode=INCLUSIVE, EndValue:"G", EndMode:EXCLUSIVE to narrow the listing to names starting with D, E, or F.

The start point of a range must always precede or be equal to the end point. Cloud Directory will return an error if EndValue precedes the StartValue. The values must also be of the same primitive type as the attribute they are filtering, String values for a String attribute, Integer for an Integer attribute, and so on. StartValue="D", StartMode=EXCLUSIVE, EndValue="D", EndMode=INCLUSIVE is invalid, for instance, because the end point includes the value while the start point follows the value.

There are three special modes that can be used by either start or end points. The following modes do not require the corresponding value field to be specified, as they imply a position on their own.

- **FIRST -** precedes all possible values in the domain. When used for the start point, this matches all possible values from the beginning of the domain up to the end point. When used for the end point, no values in the domain will match the range.
- **LAST -** follows all possible values in the domain. When used for the end point, this matches all possible values that follow the start point, including missing values. When used for the start point, no values in the domain will match the range.
- **LAST_BEFORE_MISSING_VALUES -** This mode is only useful for optional attributes where the value may be omitted (see Missing values (p. 34)). It corresponds to the point between the missing values and the actual domain values. When used for the end point, this matches all non-missing domain values that follow the start point. When used for the start point, it excludes all non-missing domain values. If the attribute is a required one, this mode is equivalent to LAST, as there can be no missing values.

Multiple range limitations

Cloud Directory limits patterns where there are multiple attributes in order to guarantee efficient, low-latency request processing. Each link with multiple identifying attributes specifies them in a well-defined

order. For instance, the Role example above defines the RoleType attribute as most significant, and the Authorizer attribute as least significant. A List request can specify only a single "qualifying" range that is not either 1) a single value or 2) spans all possible values (there can be multiple ranges that match these two requirements). Any ranges for more significant attributes than the qualifying range attribute must specify a single value, and any ranges for less significant ranges must span all possible values. In the Role example, the filter sets (RoleType:"Admin", Authorizer:["J" to "L"]) (single value + qualifying range), (RoleType:["Admin" to "User"]) (qualifying range + implicit spanning range), and (RoleType:[FIRST to LAST]) (two spanning ranges, one implicit) are all examples of valid filter sets. (RoleType:[FIRST to LAST], Authorizer:"Julia") is not a valid set, since the spanning range is more significant than the single value range.

Some useful patterns when filling in the range structures, include:

Matching a single value

Specify the value for both StartValue and EndValue, and set both modes to "INCLUSIVE".

Example: `StartValue="Admin", StartMode=INCLUSIVE, EndValue="Admin", EndMode=INCLUSIVE`

Matching a prefix

Specify the prefix as the StartValue with INCLUSIVE mode, and the first value after the prefix as EndValue with an EXCLUSIVE mode.

Example: `StartValue="Jo", StartMode=INCLUSIVE, EndValue="Jp", EndMode=EXCLUSIVE ("p" is the next character value after "o")`

Filtering for Greater Than a value

Specify the value for StartValue with EXCLUSIVE mode, and LAST as the EndMode (or LAST_BEFORE_MISSING_VALUES to exclude missing values, if applicable).

Example: `StartValue=127, StartMode=EXCLUSIVE, EndValue=null, EndMode=LAST`

Filtering for Less Than or equal to a value

Specify the value for the EndValue with INCLUSIVE mode, and FIRST as the StartMode.

Missing values

When an attribute is marked as Optional in the schema, it's value may be "missing" since it need not have been supplied when the facet was attached or the attribute could have been subsequently deleted. If the object with such a missing value is attached to an index, the index link is still present, but moved to the end of the set of links. A `ListIndex` call will first return any links where the indexed attributes are all present before returning links where one or more are missing. This is roughly similar to a relational database NULL value, with these values ordered after non-NULL values. You can specify whether a range includes these missing values or not by choosing the LAST or LAST_BEFORE_MISSING_VALUES modes. For example, you provide a filter to a ListIndex call to return just the missing values in an index by filtering with the range [LAST_BEFORE_MISSING_VALUES to LAST].

Access Objects

Objects in a directory can be accessed either by path or by `objectIdentifier`.

Path – Every object in a Cloud Directory tree can be identified and found by the pathname that describes how to reach it. The path starts from the root of the directory (Node `000` in the previous figure). The path notation begins with the link labeled with a slash (/) and follows the child links separated by path separator (also a slash) until reaching the last part of the path. For example, object `005` in the previous figure can be identified using the path `/group/a/d`. Multiple paths may identify an object, since objects that are leaf nodes can have multiple parents. The following path can also be used to identify object `005` : /group/b/e

ObjectIdentifier – Every object in the directory has a unique global identifier, which is the `ObjectIdentifier`. `ObjectIdentifier` is returned as part of the `CreateObject` API call. You can also fetch the `ObjectIdentifier` by using `GetObjectInformation` API call. For example, to fetch the object identifier of object `005`, you can call `GetObjectInformation` with object reference as the path that leads to the object, which is group/b/e or group/a/d.

```
GetObjectInformationRequest request = new GetObjectInformationRequest()
    .withDirectoryArn(directoryArn)
    .withObjectReference("/group/b/e")
    .withConsistencyLevel(level)
GetObjectInformationResult result = cdClient.getObjectInformation(request)
String objectIdentifier = result.getObjectIdentifier()
```

Populating Objects

New facets can be added to an object using `AddFacetToObject` API call. The type of the object is determined based on the facets attached to the object. Object attachment in a directory works based on the type of the object. When attaching an object, remember these rules:

- A leaf node object cannot have children.
- A node object can have multiple children.
- An object of the policy type cannot have children and can have zero or one parent.

Updating Objects

You can update an object in multiple ways:

1. Use the `UpdateObjectAttributes` operation to update individual facet attributes on an object.
2. Use the `AddFacetToObject` operation to add new facets to an object.
3. Use the `RemoveFacetFromObject` operation to delete existing facets from an object.

Deleting Objects

An attached object must meet certain conditions before you can delete it from a directory:

1. You must detach the object from the tree. You can detach an object only when it doesn't have any children. If the object has children, you must detach all the children first.
2. You can delete a detached object only if all the attributes on that object are deleted. You can delete attributes on an object by deleting each facet attached to that object. You can fetch a list of facets attached to an object by calling `GetObjectInformation`.
3. An object must also have no parent, no policy attachments, and no index attachments.

Because an object must be fully detached from the tree to be deleted, you must use the object identifier to delete it.

Querying Objects

This section discusses various elements relevant for querying objects in a directory.

Directory Traversal

Because Cloud Directory is a tree, you can query objects from the top down using the `ListObjectChildren` API operation or from the bottom up using the `ListObjectParents` API operation.

Policy Lookup

Given an object reference, the `LookupPolicy` API operation returns all the policies that are attached along its path (or paths) to the root in a top-down fashion. Any of the paths that are not leading up to the root are ignored. All policy type objects are returned.

If the object is a leaf node, it can have multiple paths to the root. This call returns only one path for each call. To fetch additional paths, use the pagination token.

Index Querying

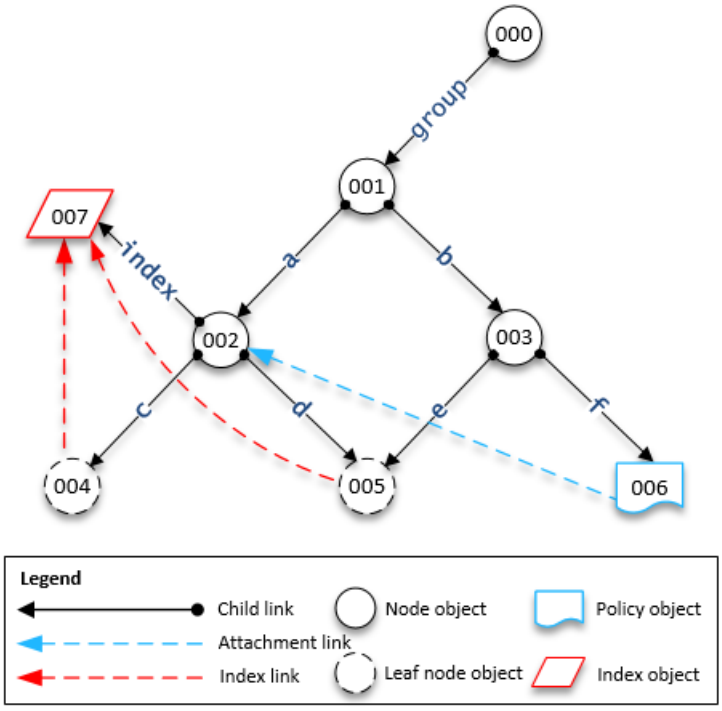
Cloud Directory supports rich index querying functionality with the use of the following ranges:

- **FIRST -** Starts from the first indexed attribute value. The start attribute value is optional.
- **LAST -** Returns attribute values up to the end of the index, including the missing values. The end attribute value is optional.
- **LAST_BEFORE_MISSING_VALUES -** Returns attribute values up to the end of index, excluding missing values.
- **INCLUSIVE -** Includes the attribute value being specified.
- **EXCLUSIVE -** Excludes the attribute value being specified.

Parent Path Listing

Using the `ListObjectParentPaths` API call, you can retrieve all available parent paths for any type of object (node, leaf node, policy node, index node). This API operation can be helpful when you need to evaluate all parents for an object. The call returns all the objects from the directory root until the requested object. It also returns the number of paths based on user-defined `MaxResults`, in case of multiple paths to the parent. The order of the paths and nodes returned is consistent among multiple API calls unless the objects are deleted or moved. Paths not leading to the directory root are ignored from the target object.

For an example on how this works, let's say a directory has an object hierarchy similar to the illustration shown below.



The numbered shapes represent the different objects. The number of arrows between that object and the directory root (`000`) represent the complete path and would be expressed in the output. The following table shows requests and responses from queries made to specific leaf node objects in the hierarchy.

Example queries on objects

Request	Response
`004, PageToken: null, MaxResults: 1`	`[{/group/a/c], [000, 001, 002, 004]}], PageToken: null`
`005, PageToken: null, MaxResults: 2`	`[{/group/a/d, [000, 001, 002, 005]}, { /group/b/e, [000, 001, 003, 005]}], PageToken: null` **Note** In this example, object `005` has both nodes `002` and `003` as parents. Also, since `MaxResults` is 2, both paths display objects in a list.
`005, PageToken: null, MaxResults: 1`	`[{/group/a/d, [000, 001, 002, 005]}], PageToken: <encrypted_next_token>`
`005, PageToken: <encrypted_next_token> MaxResults: 1`	`[{/group/b/e, [000, 001, 003, 005]}], PageToken: null` **Note** In this example, object `005` has both nodes `002` and `003` as parents. Also, since `MaxResults` is 1, multiple paginated calls with page tokens will be made to get all paths with a list of objects.
`006, PageToken: null, MaxResults: 1`	`[{/group/b/f, [000, 001, 003, 006]}], PageToken: null`

Request	Response
`007, PageToken: null, MaxResults: 1`	`[{/group/a/index, [000, 001, 002, 007]}], PageToken: null`

Consistency Levels

Amazon Cloud Directory is a distributed directory store. Data is distributed to multiple servers in different Availability Zones. A successful write request updates the data on all servers. Data is eventually available on all servers, usually within a second. To aid users of the service, Cloud Directory offers two consistency levels for read operations. This section describes the different consistency levels and eventually consistent nature of Cloud Directory.

Read Isolation Levels

When reading data from Cloud Directory, you must specify the isolation level you want to read from. Different isolation levels have tradeoffs between latency and data freshness.

- **EVENTUAL** – The snapshot isolation level reads whatever data is immediately available**.** It provides the lowest latency of any isolation level. It also provides a potentially old view of the data in the directory. EVENTUAL isolation does not provide read-after-write consistency. This means it is not guaranteed that you will be able to read data immediately after writing it.
- **SERIALIZABLE** – The serializable isolation level provides the highest level of consistency offered by Cloud Directory. Reads done at the SERIALIZABLE isolation level ensure that you receive data from any successful writes. If a change has been made to the data that you requested and that change is not yet available, the system rejects your request with `RetryableConflictException`. We recommend that you retry these exceptions (see the following section). When successfully retried, SERIALIZABLE reads offer read-after-write consistency.

Write Requests

Cloud Directory ensures that multiple write requests are not concurrently updating the same object or objects. If two write requests are found to be operating on the same objects, one of the operations fails with a `RetryableConflictException`. We recommend that you retry these exceptions (see the section below).

> **Note**
> `RetryableConflictException` responses received during write operations cannot be used to detect race conditions. Given a use case that has been shown to precipitate this situation, there is no guarantee that an exception will always occur. Whether an exception occurs or not depends on the order of each request being processed internally.

RetryableConflictExceptions

When performing write operations or read operations with a SERIALIZABLE isolation level after a write on the same object, Cloud Directory may respond with a `RetryableConflictException`. This exception indicates that Cloud Directory servers have not yet processed the contents of the previous write. These situations are transient and remedy themselves quickly. It's important to note that the `RetryableConflictException` cannot be used to detect any type of read-after-write consistency. There is no guarantee a particular use case will cause this exception.

We recommend that you configure your Cloud Directory clients to retry the `RetryableConflictException`. This configuration provides error-free behavior during operation. The following sample code demonstrates how this configuration can be made in Java.

```
     RetryPolicy retryPolicy = new RetryPolicy(new CloudDirectoryRetryCondition(),
         PredefinedRetryPolicies.DEFAULT_BACKOFF_STRATEGY,
         PredefinedRetryPolicies.DEFAULT_MAX_ERROR_RETRY,
               true);

       ClientConfiguration clientConfiguration = new
ClientConfiguration().withRetryPolicy(retryPolicy);


 AmazonCloudDirectory client = new AmazonCloudDirectory (
   new BasicAWSCredentials(…), clientConfiguration);


public static class CloudDirectoryRetryCondition extends SDKDefaultRetryCondition {

 @Override
 public boolean shouldRetry(AmazonWebServiceRequest originalRequest, AmazonClientException
exception,
   int retriesAttempted) {

  if (exception.getCause() instanceof RetryableConflictException) {
   return true;
  }

  return super.shouldRetry(originalRequest, exception, retriesAttempted);
 }
}
```

Indexing and Search

Amazon Cloud Directory supports two methods of indexing: Value based and type based. Value-based indexing is the most common form. With it you can index and search for objects in the directory based on the values of object attributes. With type-based indexing, you can index and search for objects in the directory based on object types. Facets help define object types. For more information about schemas and facets, see Schemas (p. 8) and Facets (p. 10).

Indexes in Cloud Directory enable simple listing of other objects by those objects' attribute or facet values. Each index is defined at creation to work with a specific named attribute or facet. For example, an index may be defined on the “email” attribute of the “Person” facet. Indexes are first-class objects, which means that clients can create, modify, list, and delete them flexibly according to the application logic’s needs.

Conceptually, indexes are similar to nodes with children, where the links to the indexed nodes are labeled according to the indexed attributes, rather than being given a label when the child is attached. However, index links are not parent-child edges, and have their own set of enumeration API operations.

It’s important to understand that indexes in Cloud Directory are not automatically populated as they might be in other systems. Instead, you use API calls to directly attach and detach objects to or from the index. Although this is a bit more work, it gives you flexibility to define varying index scopes. For example, you can define an index that tracks only direct children of a specific node. Or you can define an index that tracks all objects in a given branch under a local root, such as all nodes in a department. You can also do both at the same time.

Topics

Index Lifecycle

You can use the following API calls to help with the development lifecycle of indexes.

1. You create indexes with the `CreateIndex` API call. You supply an index definition structure that describes the attributes on attached objects that the index will track. The definition also indicates whether or not the index should enforce uniqueness. The result is an object ID for the new index, which should immediately be attached to your hierarchy like any other object. For example, this can be a branch dedicated to holding indexes.
2. You attach objects to the index manually with the `AttachToIndex` API call. The index then automatically tracks the values of its defined attributes on each attached object.
3. To use the indexes to search for objects with more efficient enumeration, call `ListIndex` and specify a range of values that you are interested in.
4. Use the `ListAttachedIndices` API call to enumerate the indexes that are attached to a given object.
5. Use the `DetachFromIndex` API call to remove objects from the index manually.
6. Once you detach all objects from the index, you can delete the index with the `DeleteObject` API call.

There is no limit on the number of indexes within a directory, other than the limit on the space used by all objects. Indexes and their attachments do consume space, but it is similar to that consumed by nodes

and parent–child links. There is a limit on the number of indexes that can be attached to a given object. For more information, see Amazon Cloud Directory Limits (p. 64).

Facet-Based Indexing

With facet-based indexing and search, you can optimize your directory searches by searching only a subset of your directory. To do this, you use a schema *facet*. For example, instead of searching across all the user objects in your directory, you can search only the user objects that contain an employee facet. This efficiency helps reduce the latency time and amount of data retrieved for the query.

With facet-based indexing, you can use the Cloud Directory index API operations to create and attach an index to the facets of objects. You can also list the index results and then filter those results based on certain facets. This can effectively reduce query times and the amount of data by narrowing the search scope to only objects that contain a certain type of facet.

The `"facets"` attribute that is used with the `CreateIndex` and `ListIndex` API calls surfaces the collection of facets that are applied to an object. This attribute is available for use only with the `CreateIndex` and `ListIndex` API calls. As shown in the following sample code, the schema ARN uses the directory's region, owner account, and directory ID to reference the Cloud Directory schema. This service-provided schema does not appear in listings.

```
String cloudDirectorySchemaArn = String.format("arn:aws:clouddirectory:%s:%s:directory/%s/
schema/CloudDirectory/1.0", region, ownerAccount, directoryId);
```

For example, the following sample code creates a facet-based index specific to your AWS account and directory where you could enumerate all objects that are created with the facet `SalesDepartmentFacet`.

Note
Make sure to use the "facets" value within the parameters as shown below. Instances of "facets" shown in the sample code refer to a value that is provided and controlled by the Cloud Directory service. You can use these for indexing but can have read-only access.

```
// Create a facet-based index
String cloudDirectorySchemaArn = String.format("arn:aws:clouddirectory:%s:%s:directory/%s/
schema/CloudDirectory/1.0",
    region, ownerAccount, directoryId);

facetIndexResult = clouddirectoryClient.createIndex(new CreateIndexRequest()
  .withDirectoryArn(directoryArn)
  .withOrderedIndexedAttributeList(List(new AttributeKey()
        .withSchemaArn(cloudDirectorySchemaArn)
        .withFacetName("facets")
        .withName("facets")))
        .withIsUnique(false)
        .withParentReference("/")
        .withLinkName("MyFirstFacetIndex"))
facetIndex = facetIndexResult.getObjectIdentifier()

// Attach objects to the facet-based index
clouddirectoryClient.attachToIndex(new
 AttachToIndexRequest().withDirectoryArn(directoryArn)
  .withIndexReference(facetIndex).withTargetReference(userObj))

// List all objects
val listResults = clouddirectoryClient.listIndex(new ListIndexRequest()
  .withDirectoryArn(directoryArn)
  .withIndexReference(facetIndex)
  .getIndexAttachments()
```

```
// List the index results filtering for a certain facet
val filteredResults = clouddirectoryClient.listIndex(new ListIndexRequest()
  .withDirectoryArn(directoryArn)
  .withIndexReference(facetIndex)
  .withRangesOnIndexedValues(new ObjectAttributeRange()
    .withAttributeKey(new AttributeKey()
      .withFacetName("facets")
      .withName("facets")
      .withSchemaArn(cloudDirectorySchemaArn))
    .withRange(new TypedAttributeValueRange()
      .withStartMode(RangeMode.INCLUSIVE)
      .withStartValue("MySchema/1.0/SalesDepartmentFacet")
      .withEndMode(RangeMode.INCLUSIVE)
      .withEndValue("MySchema/1.0/SalesDepartmentFacet")
    )))
```

Unique vs Nonunique Indexes

Unique indexes differ from nonunique indexes in enforcing uniqueness of the indexed attribute values for objects that are attached to the index. For example, you might want to populate Person objects into two indexes, a unique one on an “email” attribute, and a nonunique one on a “lastname” attribute. The lastname index allows many Person objects with the same last name to be attached. On the other hand, the `AttachToIndex` call that targets the email index returns a `LinkNameAlreadyInUseException` error if a Person with the same email attribute is already attached. Note that the error does not remove the Person object itself. Consequently, an application might create the Person, attach it to the hierarchy, and attach it to indexes, all in a single batch request. This ensures that if uniqueness is violated on any of the indexes, the object and all of its attachments are rolled back automatically.

How To Administer Cloud Directory

This section lists all of the procedures for operating and maintaining a Cloud Directory environment.

Topics

Manage Your Directories

This section describes how to maintain common directory tasks for your Cloud Directory environment.

Topics

Create Your Directory

Before you can create a directory in Amazon Cloud Directory, AWS Directory Service requires that you first apply a schema to it. A directory cannot be created without a schema and typically has one schema applied to it. However, you use Cloud Directory API operations to apply additional schemas to a directory. For more information, see ApplySchema in the *Amazon Cloud Directory API Reference Guide*.

To create a Cloud Directory

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Directories**.
2. Choose **Set up Cloud Directory**.
3. Under **Choose a schema to apply to your new directory**, type the friendly name of your directory, such as `User Repository`, and then choose one of the following options:

   - **Managed schema**
   - **Sample schema**
   - **Custom schema**

   Sample schemas and custom schemas are placed in the **Development** state, by default. For more information about schema states, see Schema Lifecycle (p. 8). Before a schema can be applied to a directory, it must be converted into the **Published** state. To successfully publish a sample schema using the console, you must have permissions to the following actions:

   - `clouddirectory:Get*`
   - `clouddirectory:List*`
   - `clouddirectory:CreateSchema`
   - `clouddirectory:CreateDirectory`

- `clouddirectory:PutSchemaFromJson`
- `clouddirectory:PublishSchema`
- `clouddirectory:DeleteSchema`

Since sample schemas are read-only templates provided by AWS, they cannot be published directly. Instead, when you choose to create a directory based on a sample schema, the console creates a temporary copy of the sample schema you selected and places it in the **Development** state. It then creates a copy of that development schema and places it in the **Published** state. Once published, the development schema is deleted, which is why the `DeleteSchema` action is necessary when publishing a sample schema.

4. Choose **Next**.
5. Review the directory information and make any necessary changes. When the information is correct, choose **Create**.

Delete Your Directory

Use the following procedure to delete a directory in Cloud Directory.

Note
Before you can delete a directory, you must first disable it. For instructions, see Disable Your Directory (p. 44).

To delete a directory

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Directories**.
2. Select the option in the table next to the directory ID you want to delete.
3. Choose **Actions**.
4. Choose **Delete**
5. In the **Delete directory** dialog, confirm the operation by typing in the name of your directory, and then choose **Delete**.

Disable Your Directory

Use the following procedure to disable a directory in Cloud Directory.

To disable a directory

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Directories**.
2. Select the option in the table next to the directory ID you want to disable.
3. Choose **Actions**.
4. Choose **Disable**

Enable Your Directory

Use the following procedure to enable a previously disabled directory in Cloud Directory.

To enable a directory

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Directories**.

2. Select the option in the table next to the directory ID you want to enable.
3. Choose **Actions**.
4. Choose **Enable**

Manage Your Schema

This section describes how to maintain common schema tasks for your Cloud Directory environment.

Topics

Create Your Schema

Amazon Cloud Directory supports uploading of a compliant JSON file for schema creation. To create a new schema, you can either create your own JSON file from scratch or download one of the existing schemas listed in the console. Then upload it as a custom schema. For more information, see Custom Schemas (p. 17).

You can also create, delete, download, list, publish, update and upgrade schemas using the Cloud Directory APIs. For more information about schema API operations, see the Amazon Cloud Directory API Reference Guide.

Choose either of the procedures below, depending on your preferred method.

To create a custom schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Schemas**.
2. Create a JSON file with all of your new schema definitions. For more information about how to format a JSON file, see JSON Schema Format (p. 21).
3. In the console, choose **Upload new schema**.
4. In the **Upload new schema** dialog, type a name for the schema.
5. Select **Choose file**, select the new JSON file that you just created, and then choose **Open**.
6. Choose **Upload**. This adds a new schema to your schema library and places it in the **Development** state. For more information about schema states, see Schema Lifecycle (p. 8).

To create a custom schema based on an existing one in the console

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, choose **Schemas**.
2. In the table listing the schemas, select the option near the schema you want to copy.
3. Choose **Actions**.
4. Choose **Download schema**.
5. Rename the JSON file, edit it as needed, and then save the file. For more information about how to format a JSON file, see JSON Schema Format (p. 21).

6. In the console, choose **Upload new schema**, select the JSON file that you just edited, and then choose **Open**.

   This adds a new schema to your schema library and places it in the **Development** state. For more information about schema states, see Schema Lifecycle (p. 8).

Delete a Schema

Use the following procedure to delete a schema in Cloud Directory.

To delete a schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Schemas**.
2. Select the option in the table next to the schema name you want to delete.
3. Choose **Actions**.
4. Choose **Delete**
5. In the **Delete schema** dialog, confirm the operation by choosing **Delete**.

Download a Schema

Use the following procedure to download a schema.

To download a schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Schemas**.
2. Select the option in the table next to the schema name you want to download.
3. Choose **Actions**.
4. Choose **Download schema**

Publish a Schema

Use the following procedure to publish a schema in Cloud Directory.

To publish a schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Schemas**.
2. Select the option in the table next to the schema name you want to publish.
3. Choose **Actions**.
4. Choose **Publish**
5. In the **Publish schema** dialog, provide the following information:

   a. **Schema name**
   b. **Major version**
   c. **Minor version**
6. Choose **Publish**.

Update Your Schema

Use the following procedure to update a schema in Cloud Directory.

To update a schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Schemas**.
2. Select the option in the table next to the schema name you want to update.
3. Choose **Actions**.
4. Choose **Update**
5. In the **Update schema** dialog, optionally modify the **Schema name**, or select **Choose file** to apply or remove facets and attributes.
6. Choose **Update**.

Upgrade Your Schema

Upgrading a schema will add the facets and attributes you choose to the published schema you select. Use the following procedure to upgrade a published schema.

To upgrade a schema

1. In the AWS Directory Service console navigation pane, under **Cloud Directory**, select **Schemas**.
2. Select the option in the table next to the schema name you want to upgrade.
3. Choose **Actions**.
4. Choose **Upgrade**
5. In the **Upgrade published schema** dialog, choose either of the following options and then choose **Upgrade**:
   - **Choose from your current list of development schemas**
   - **Upload a new schema file (JSON)**
6. Choose **upgrade**.

Authentication and Access Control for Amazon Cloud Directory

Access to Amazon Cloud Directory requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Cloud Directory to help secure your resources by controlling who can access them:

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create a directory in Cloud Directory). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Cloud Directory supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

- **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Cloud Directory resources. For example, you must have permissions to create an Amazon Cloud Directory.

The following sections describe how to manage permissions for Cloud Directory. We recommend that you read the overview first.

Overview of Managing Access Permissions to Your Cloud Directory Resources

Every AWS resource is owned by an AWS account, and permissions to create or access the resources are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

Cloud Directory Resources and Operations

In Cloud Directory, the primary resources are directories and schemas. These resources have unique Amazon Resource Names (ARNs) associated with them as shown in the following table.

Resource Type	ARN Format
Directory	`arn:aws:clouddirectory:region:account-id:directory/directory-id`
Schema	`arn:aws:clouddirectory:region:account-id:schema/schema-state/schema-name`

For more information about schema states and ARNs, see ARN Examples in the *Amazon Cloud Directory API Reference*.

Cloud Directory provides a set of operations to work with the appropriate resources. For a list of available operations, see either Amazon Cloud Directory Actions or Directory Service Actions.

Understanding Resource Ownership

A *resource owner* is the AWS account that created a resource. That is, the resource owner is the AWS account of the *principal entity* (the root account, an IAM user, or an IAM role) that authenticates the request that creates the resource. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a Cloud Directory resource, such as a directory, your AWS account is the owner of that resource.
- If you create an IAM user in your AWS account and grant permissions to create Cloud Directory resources to that user, the user can also create Cloud Directory resources. However, your AWS account, to which the user belongs, owns the resources.
- If you create an IAM role in your AWS account with permissions to create Cloud Directory resources, anyone who can assume the role can create Cloud Directory resources. Your AWS account, to which the role belongs, owns the Cloud Directory resources.

Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

> **Note**
> This section discusses using IAM in the context of Cloud Directory. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM polices) and policies attached to a resource are referred to as *resource-based* policies. Cloud Directory supports only identity-based policies (IAM policies).

Topics

Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – An account administrator can use a permissions policy that is associated with a particular user to grant permissions for that user to create an Cloud Directory resource, such as a new directory.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following permissions policy grants permissions to a user to run all of the actions that begin with `Create`. These actions show information about an Cloud Directory resource, such as a directory or schema. Note that the wildcard character (*) in the `Resource` element indicates that the actions are allowed for all Cloud Directory resources owned by the account.

```
{
   "Version":"2017-01-11",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"clouddirectory:Create*",
         "Resource":"*"
      }
   ]
}
```

For more information about using identity-based policies with Cloud Directory, see Using Identity-Based Policies (IAM Policies) for Cloud Directory (p. 52). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Cloud Directory doesn't support resource-based policies.

Specifying Policy Elements: Actions, Effects, Resources, and Principals

For each Cloud Directory resource (see Cloud Directory Resources and Operations (p. 50)), the service defines a set of API operations. For a list of available API operations, see either Amazon Cloud Directory

Actions or Directory Service Actions. To grant permissions for these API operations, Cloud Directory defines a set of actions that you can specify in a policy. Note that, performing an API operation can require permissions for more than one action.

The following are the basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies. For Cloud Directory resources, you always use the wildcard character (*) in IAM policies. For more information, see Cloud Directory Resources and Operations (p. 50).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `clouddirectory:GetDirectory` permission allows the user permissions to perform the Cloud Directory `GetDirectory` operation.
- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). Cloud Directory doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the Amazon Cloud Directory API actions and the resources that they apply to, see Amazon Cloud Directory API Permissions: Actions, Resources, and Conditions Reference (p. 53).

Specifying Conditions in a Policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Cloud Directory. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Global Condition Keys in the *IAM User Guide*.

Using Identity-Based Policies (IAM Policies) for Cloud Directory

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

> **Important**
> We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Cloud Directory resources. For more information, see Overview of Managing Access Permissions to Your Cloud Directory Resources (p. 49).

The sections in this topic cover the following:

- Permissions Required to Use the AWS Directory Service Console (p. 53)

Permissions Required to Use the AWS Directory Service Console

For a user to work with the AWS Directory Service console, that user must have permissions listed in the policy above or the permissions granted by the Directory Service Full Access Role or Directory Service Read Only role, described in AWS Managed (Predefined) Policies for Amazon Cloud Directory (p. 53).

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy.

AWS Managed (Predefined) Policies for Amazon Cloud Directory

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed policies grant necessary permissions for common use cases so you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Cloud Directory:

- **AmazonCloudDirectoryReadOnlyAccess** – Grants a user or group read-only access to all Amazon Cloud Directory resources. For more information, see the Policies page in the AWS Management Console.
- **AmazonCloudDirectoryFullAccess** – Grants a user or group full access to Amazon Cloud Directory. For more information, see the Policies page in the AWS Management Console.

In addition, there are other AWS-managed policies that are suitable for use with other IAM roles. These policies are assigned to the roles associated with users in your Amazon Cloud Directory and are required in order for those users to have access to other AWS resources, such as Amazon EC2.

You can also create custom IAM policies that allow users to access the required API actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Amazon Cloud Directory API Permissions: Actions, Resources, and Conditions Reference

When you are setting up Access Control (p. 49) and writing permissions policies that you can attach to an IAM identity (identity-based policies), you can use the following table as a reference. The list includes each Amazon Cloud Directory API operation, the corresponding actions for which you can grant permissions to perform the action, the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field and the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your Amazon Cloud Directory policies to express conditions. For a complete list of AWS-wide keys, see Available Global Condition Keys in the *IAM User Guide*.

> **Note**
> To specify an action, use the `clouddirectory:` prefix followed by the API operation name (for example, `clouddirectory:CreateDirectory`).

Transaction Support

With Amazon Cloud Directory, it's often necessary to add new objects or add relationships between new objects and existing objects to reflect changes in a real-world hierarchy. Batch operations can make directory tasks like these easier to manage by providing the following benefits:

- Batch operations can minimize the number of round trips required to write and read objects to and from your directory, improving the overall performance of your application.
- Batch write provides the SQL database-equivalent transaction semantics. All operations successfully complete, or if any operation has a failure then none of them are applied.
- Using batch reference you can create an object and use a reference to the new object for further action such as adding it to a relationship, reducing overhead of using a read operation before a write operation.

BatchWrite

Use BatchWrite operations to perform multiple write operations on a directory. All operations in batch write are executed sequentially. It works similar to SQL database transactions. If one of the operation inside batch write fails, the entire batch write has no effect on the directory. If a batch write fails, a batch write exception occurs. The exception contains the index of the operation that failed along with exception type and message. This information can help you identify the root cause for the failure.

The following API operations are supported as part of batch write:

- AddFacetToObject
- AttachObject
- AttachPolicy
- AttachToIndex
- AttachTypedLink
- CreateIndex
- CreateObject
- DeleteObject
- DetachFromIndex
- DetachObject
- DetachTypedLink
- RemoveFacetFromObject
- UpdateObjectAttributes

Batch Reference Name

Batch reference names are supported only for batch writes when you need to refer to an object as part of the intermediate batch operation. For example, suppose that as part of a given batch write, 10 different objects are being detached and are attached to a different part of the directory. Without batch reference, you would have to read all 10 object references and provide them as input during

reattachment as part of the batch write. You can use a batch reference to identify the detached resource during attachment. A batch reference can be any regular string prefixed with the number sign / hashtag symbol (#).

For example, in the following code sample, an object with link name `"this-is-a-typo"` is being detached from root with a batch reference name `"ref"` . Later the same object is attached to the root with the link name as `"correct-link-name"` . The object is identified with the child reference set to batch reference. Without the batch reference, you would initially need to get the `objectIdentifier` that is being detached and provide that in the child reference during attachment. You can use a batch reference name to avoid this extra read.

```
BatchDetachObject batchDetach = new BatchDetachObject()
        .withBatchReferenceName("ref")
        .withLinkName("this-is-a-typo")
        .withParentReference(new ObjectReference().withSelector("/"));
BatchAttachObject batchAttach = new BatchAttachObject()
        .withParentReference(new ObjectReference().withSelector("/"))
        .withChildReference(new ObjectReference().withSelector("#ref"))
        .withLinkName("correct-link-name");
BatchWriteRequest batchWrite = new BatchWriteRequest()
        .withDirectoryArn(directoryArn)
        .withOperations(new ArrayList(Arrays.asList(batchDetach, batchAttach)));
```

BatchRead

Use BatchRead operations to perform multiple read operations on a directory. For example, in the following code sample, children of object with reference `"/managers"` is being read along with attributes of object with reference `"/managers/bob"` in a single batch read.

```
BatchListObjectChildren listObjectChildrenRequest = new BatchListObjectChildren()
        .withObjectReference(new ObjectReference().withSelector("/managers"));
BatchListObjectAttributes listObjectAttributesRequest = new BatchListObjectAttributes()
        .withObjectReference(new ObjectReference().withSelector("/managers/bob"));
BatchReadRequest batchRead = new BatchReadRequest()
        .withConsistencyLevel(ConsistencyLevel.SERIALIZABLE)
        .withDirectoryArn(directoryArn)
        .withOperations(new ArrayList(Arrays.asList(listObjectChildrenRequest,
 listObjectAttributesRequest)));
BatchReadResult result = cloudDirectoryClient.batchRead(batchRead);
```

BatchRead supports the following API operations:

- GetObjectInformation
- ListAttachedIndices
- ListIncomingTypedLinks
- ListIndex
- ListObjectAttributes
- ListObjectChildren
- ListObjectParentPaths
- ListObjectPolicies
- ListOutgoingTypedLinks
- ListPolicyAttachments
- LookupPolicy

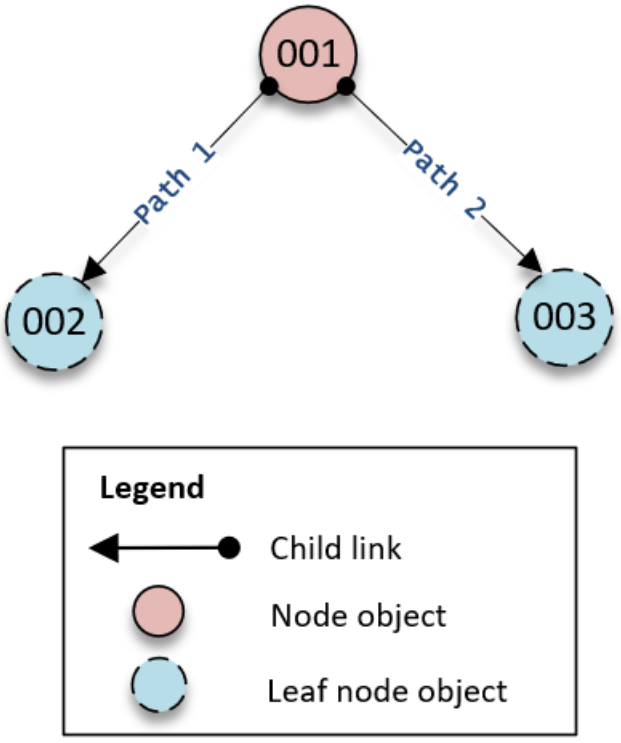
Limits on Batch operations

Each request to the server (including batched requests) has a maximum number of resources that can be operated on, regardless of the number of operations in the request. This allows you to compose batch requests with high flexibility as long as you stay within the resource maximums. For more information on resource maximums, see Amazon Cloud Directory Limits (p. 64).

Limits are calculated by summing the writes or reads for each single operation inside the Batch. For example, the read operation limit is currently 200 objects per API call. Let's say you want to compose a batch that adds 9 ListObjectChildren API calls and each call requires reading 20 objects . Since the total number of read objects (9 x 20 = 180) does not exceed 200, the batch operation would succeed.

The same concept applies with calculating write operations. For example, the write operation limit is currently 20. If you set up your batch to add 2 UpdateObjectAttributes API calls with 9 write operations each, this would also succeed. In either case, should the batch operation exceed the limit, then the operation will fail and a `LimitExceededException` will be thrown.

The correct way to calculate the number of objects that are included within a batch is to include both the actual node or leaf_node objects and if using a path based approach to iterate your directory tree, you also need to include each path that is iterated on, within the batch. For example, as shown in the following illustration of a basic directory tree, to read an attribute value for the object `003`, the total read count of objects would be three.



The traversing of reads down the tree works like this:

1. Read object `001` object to determine the path to object `003`

2. Go down `Path 2`

3. Read object `003`

Similarly, for the number of attributes we need to count the number of attributes in objects `001` and `003` to ensure we don't hit the limit.

Exception handling

Batch operations in Cloud Directory can sometimes fail. In these cases, it is important to know how to handle such failures. The method you use to resolve failures differs for write operations and read operations.

Batch write operation failures

If a batch write operation fails, Cloud Directory fails the entire batch operation and returns an exception. The exception contains the index of the operation that failed along with the exception type and message. If you see `RetryableConflictException`, you can try again with exponential backoff. A simple way to do this is to double the amount of time you wait each time you get an exception or failure. For example, if your first batch write operation fails, wait 100 milliseconds and try the request again. If the second request fails, wait 200 milliseconds and try again. If the third request fails, wait 400 milliseconds and try again.

Batch read operation failures

If a batch read operation fails, the response contains either a successful response or an exception response. Individual batch read operation failures do not cause the entire batch read operation to fail—Cloud Directory returns individual success or failure responses for each operation.

Related Cloud Directory Blog Articles

- Write and Read Multiple Objects in Amazon Cloud Directory by Using Batch Operations
- How to Use Batch References in Amazon Cloud Directory to Refer to New Objects in a Batch Request

Amazon Cloud Directory Compliance

Amazon Cloud Directory has undergone auditing for the following standards and can be part of your solution when you need to obtain compliance certification.

FedRAMP	Amazon Cloud Directory meets Federal Risk and Authorization Management Program (FedRAMP) security requirements and has received a FedRAMP Joint Authorization Board (JAB) Provisional Authority to Operate (P-ATO) at the FedRAMP Moderate Baseline. For more information about FedRAMP, see FedRAMP Compliance.
PCI Security Standards Council PARTICIPATING ORGANIZATION™	Amazon Cloud Directory has an Attestation of Compliance for Payment Card Industry (PCI) Data Security Standard (DSS) version 3.2 at Service Provider Level 1. Customers who use AWS products and services to store, process, or transmit cardholder data can use Cloud Directory as they manage their own PCI DSS compliance certification. For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see PCI DSS Level 1.
HIPAA	AWS has expanded its Health Insurance Portability and Accountability Act (HIPAA) compliance program to include Amazon Cloud Directory as a HIPAA Eligible Service. If you have an executed Business Associate Agreement (BAA) with AWS, you can use Cloud Directory to help build your HIPAA-compliant applications. AWS offers a HIPAA-focused Whitepaper for customers who are interested in learning more about how they can leverage AWS for the processing and storage of health information. For more information, see HIPAA Compliance

	Amazon Cloud Directory has successfully completed compliance certification for ISO/IEC 27001, ISO/IEC 27017, ISO/IEC 27018, and ISO 9001. For more information, see ISO 27001, ISO 27017, ISO 27018, and ISO 9001.
	System and Organization Control (SOC) reports are independent third-party examination reports that demonstrate how Amazon Cloud Directory achieves key compliance controls and objectives. The purpose of these reports is to help you and your auditors understand the AWS controls that are established to support operations and compliance. For more information, see SOC Compliance.

Shared Responsibility

Security, including HIPAA and PCI compliance, is a shared responsibility. It is important to understand that Cloud Directory compliance status does not automatically apply to applications that you run in the AWS Cloud. You must ensure that your use of AWS services complies with the standards.

Using the Cloud Directory APIs

Amazon Cloud Directory includes a set of API operations that enable programmatic access to Cloud Directory capabilities. You can use the Amazon Cloud Directory API Reference Guide to learn how to make requests to the Cloud Directory API for creating and managing the various elements. It also covers the components of requests, the content of responses, and how to authenticate requests.

Cloud Directory provides all necessary API operations that enable developers to build new applications. It provides the following categories of API calls:

- Create, read, update, delete (CRUD) for schema
- CRUD for facet
- CRUD for directories
- CRUD for objects (nodes, policies, etc.)
- CRUD for Index definition
- Batch read, batch write

How Billing Works With Cloud Directory APIs

Billing for API calls vary based on the specific types of API calls being made. There are specific billing rates for Eventually Consistent Read API calls, Strongly Consistent Read API calls, and Write API calls. Metadata API calls are free.

Strongly Consistent operations are used for read-after-write consistency when reading a value. Eventually Consistent operations are used for retrieving a value while updates are running. With Eventually Consistent operations retrieved results might not be the most accurate since the specific host you are reading the value from is still processing updates. However, the latency for such read operations are low when you retrieve a performance call.

When reading data from Cloud Directory, you must specify either an Eventually Consistent Read or Strongly Consistent Read type operation. The read type is based on consistency level. The two consistency levels are EVENTUAL for Eventually Consistent Reads and SERIALIZABLE for Strongly Consistent Reads. For more information, see Consistency Levels (p. 38).

The following table lists all of the Cloud Directory APIs and how they can impact billing for your AWS account.

API	Eventually Consistent Read [1]	Strongly Consistent Read [2]	Write [3]	Metadata [4]
AddFacetToObject			X	
ApplySchema				X
AttachObject			X	
AttachPolicy			X	
AttachToIndex			X	
AttachTypedLink			X	
BatchRead	X	X		
BatchWrite			X	

API	Eventually Consistent Read [1]	Strongly Consistent Read [2]	Write [3]	Metadata [4]
CreateDirectory			X	
CreateFacet				X
CreateIndex			X	
CreateObject			X	
CreateSchema				X
CreateTypedLinkFacet				X
DeleteDirectory				X
DeleteFacet				X
DeleteObject			X	
DeleteSchema				X
DetachFromIndex			X	
DetachObject			X	
DetachPolicy			X	
DetachTypedLink			X	
DeleteTypedLinkFacet				X
DisableDirectory				X
EnableDirectory			X	
GetAppliedSchemaVersion				X
GetDirectory				X
GetFacet				X
GetLinkAttributes	X	X		
GetObjectAttributes	X	X		
GetObjectInformation	X	X		
GetSchemaAsJson				X
GetTypedLinkFacetInformation				X
ListAppliedSchemaArns				X
ListAttachedIndices	X	X		
ListDevelopmentSchemaArns				X
ListDirectories				X
ListFacetAttributes				X

API	Eventually Consistent Read [1]	Strongly Consistent Read [2]	Write [3]	Metadata [4]
ListFacetNames				X
ListIncomingTypedLinks	X	X		
ListIndex	X	X		
ListManagedSchemaArns				X
ListObjectAttributes	X	X		
ListObjectChildren	X	X		
ListObjectParentPaths	X			
ListObjectParents	X	X		
ListObjectPolicies	X	X		
ListOutgoingTypedLinks	X	X		
ListPolicyAttachments	X	X		
ListPublishedSchemaArns				X
ListTagsForResource				X
ListTypedLinkFacetAttributes				X
ListTypedLinkFacetNames				X
LookupPolicy	X			
PublishSchema				X
PutSchemaFromJson				X
RemoveFacetFromObject			X	
TagResource				X
UntagResource				X
UpdateFacet				X
UpdateLinkAttributes			X	
UpdateObjectAttributes			X	
UpdateSchema				X
UpdateTypedLinkFacet				X
UpgradeAppliedSchema				X
UpgradePublishedSchema				X

[1] Eventually Consistent Read APIs are called with the EVENTUAL consistency level

[2] Strongly Consistent Read APIs are called with the SERIALIZABLE consistency level

[3] Write APIs are billed as write API calls

[4] Metadata APIs are NOT billed but are categorized as Metadata API calls

For additional information about billing, see Amazon Cloud Directory Pricing.

Amazon Cloud Directory Limits

The following are the default limits for Cloud Directory. Each limit is per region unless otherwise noted.

Amazon Cloud Directory

Schema and Directory Limits

Limit/Concept	Quantity
Number of attributes per facet (including required)	1000
Number of facets per object	5
Number of unique indexes an object is attached	3
Number of facets per schema	30
Number of rules per attribute	5
Number of attributes with default values per facet	10
Number of required attributes per facet	30
Number of development schemas	20
Number of published schemas	20
Number of applied schemas	5
Number of directories	100
Max page elements	30
Max input size (all inputs combined)	200 KB
Max response size (all outputs combined)	1 MB
Schema JSON file size limit	200 KB
Facet name length	64 UTF-8 encoded bytes
Directory name length	64 UTF-8 encoded bytes
Schema name length	64 UTF-8 encoded bytes

Object Limits

Limit/Concept	Quantity
Number of written objects	20 per API call
Number of read objects	200 per API call

Limit/Concept	Quantity
Number of written attribute values	1000 per API call
Number of read attribute values	1000 per API call
Path depth	15
Max input size (all inputs combined)	200 KB
Max response size (all outputs combined)	1 MB
Policy size limit	10 KB
Number of attributes that can be deleted during object deletion	30
Aggregate value length for typed link identity attributes	64 UTF-8 encoded bytes
Edge or link name length	64 UTF-8 encoded bytes
Value length for indexed attributes	64 UTF-8 encoded bytes
Value length for non-indexed attributes	2KB
Number of policies attached to an object	4

Limits on batch operations

There are no limits on the number of operations you can call inside a batch. For more information, see Limits on Batch operations (p. 56).

Limits that cannot be modified

Amazon Cloud Directory limits that cannot be changed or increased, include:

- Facet name length
- Directory name length
- Schema name length
- Max page elements
- Edge or link name length
- Value length for indexed attributes

Cloud Directory Resources

The following tables list related resources that you'll find useful as you work with this service.

Cloud Directory Getting Started	Link
Cloud Directory Webinar	https://www.youtube.com/watch?v=UANm3DC_IxE
Cloud Directory Sample Java Code	https://github.com/aws-samples/AmazonCloudDirectory-sample

Cloud Directory Blog Posts	Description
How to rapidly develop applications on Amazon Cloud Directory with Managed Schema	This blog post explains about rapidly prototyping and developing on Cloud Directory using managed schema. It also includes sample Java code.
How to Search More Efficiently in Amazon Cloud Directory	This blog post explains about searching more efficiently using facet-based indexing. It also includes sample Java code.
How to Easily Apply Amazon Cloud Directory Schema Changes with In-Place Schema Upgrades	This blog post explains about performing an in-place schema upgrade for any operational (running) Cloud Directory. It also includes sample Java code.
Write and Read Multiple Objects in Amazon Cloud Directory by Using Batch Operations	Explains about using Batch Read and Write. It also includes sample Java code.
How to Use Batch References in Amazon Cloud Directory to Refer to New Objects in a Batch Request	Explains about using Batch Reference. It also includes sample Java code.
Cloud Directory Update – Support for Typed Links	Explains about creating and searching relationships across hierarchies in Cloud Directory by using typed links. It also includes sample Java code.
New Cloud Directory API Makes It Easier to Query Data Along Multiple Dimensions	Explains how to query for data across multiple dimensions with a single call using the `ListObjectParentPaths` API.
How to Create an Organizational Chart with Separate Hierarchies by Using Amazon Cloud Directory	Explains how to create a schema and a directory with sample Java code.
Amazon Cloud Directory – A Cloud-Native Directory for Hierarchical Data	Describes the launch of Cloud Directory as a new service from AWS.

Cloud Directory Documentation	Link
Cloud Directory Developer Guide	https://docs.aws.amazon.com/clouddirectory/latest/developerguide/what_is_cloud_directory.html
Cloud Directory API Reference	http://docs.aws.amazon.com/amazoncds/latest/APIReference/welcome.html
Cloud Directory Limits	https://docs.aws.amazon.com/clouddirectory/latest/developerguide/limits.html

Cloud Directory Other	Link
Cloud Directory Product Info	https://aws.amazon.com/cloud-directory/
Cloud Directory Pricing	https://aws.amazon.com/cloud-directory/pricing/

Document History

The following table describes the documentation changes since the last release of the *Amazon Cloud Directory Developer Guide*.

- **Latest documentation update:** June 21, 2018

update-history-change	update-history-description	update-history-date
New managed schema	Added content for managed schema option.	June 21, 2018
Migrated content to this guide	Transferred all existing Cloud Directory content from the AWS Directory Service Admin Guide to this new Amazon Cloud Directory Developer Guide to more directly map to customer needs.	June 20, 2018
In-place schema upgrades	Added content for applying schema changes across your Amazon Cloud Directory directories with in-place schema upgrades.	December 6, 2017
Facet-based indexing	Added facet-based index section.	August 9, 2017
Batches	Updated information about batches for Amazon Cloud Directory.	July 26, 2017
Compliance	Added information about HIPAA and PCI compliance.	July 14, 2017
Typed links	Added new typed links content for Amazon Cloud Directory.	May 31, 2017
Amazon Cloud Directory service launch	New directory type introduced.	January 26, 2017

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.