aws

Developer Guide

AWS Cloud Map

AWS Cloud Map: Developer Guide

Table of Contents

What Is AWS Cloud Map?

AWS Cloud Map is a fully managed solution that you can use to map logical names to the backend services and resources that your applications depend on. It also helps your applications discover resources using one of the AWS SDKs, RESTful API calls, or DNS queries. AWS Cloud Map serves only healthy resources, which can be Amazon DynamoDB (DynamoDB) tables, Amazon Simple Queue Service (Amazon SQS) queues, any higher-level application services that are built using Amazon Elastic Compute Cloud (Amazon EC2) instances or Amazon Elastic Container Service (Amazon ECS) tasks, and more.

Components of AWS Cloud Map

Namespace

To get started, you first create a AWS Cloud Map namespace that functions as a way to group services for an application. A namespace identifies the name that you want to use to locate your resources and also specifies how you want to locate resources: using AWS Cloud Map DiscoverInstances API calls, DNS queries in a VPC, or public DNS queries. In most cases, a namespace contains all the services for an application, such as a billing application. For more information, see AWS Cloud Map namespaces.

Service

After creating a namespace, you create an AWS Cloud Map service for each type of resource for which you want to use AWS Cloud Map to locate endpoints. For example, you might create services for web servers and database servers.

A service is a template that AWS Cloud Map uses when your application adds another resource, such as another web server. If you chose to locate resources using DNS when you created the namespace, a service contains information about the types of records that you want to use to locate the web server. A service also indicates whether you want to check the health of the resource and whether you want to use Amazon Route 53 health checks or a third-party health checker. For more information, see AWS Cloud Map services.

Service instance

When your application adds a resource, you can call the AWS Cloud Map RegisterInstance API action in the code, which creates a AWS Cloud Map service instance in a service. The service

instance contains information about how your application can locate the resource, whether using DNS or using the AWS Cloud Map DiscoverInstances API action.

When your application needs to connect to a resource, it calls DiscoverInstances or utilizes public or private DNS queries by specifying the namespace and service that are associated with the resource. AWS Cloud Map returns information about how to locate one or more resources. If you specified health checking when you created the service, AWS Cloud Map returns only healthy instances. For more information, see AWS Cloud Map service instances.

Accessing AWS Cloud Map

You can access AWS Cloud Map in the following ways:

- **AWS Management Console** – The procedures throughout this guide explain how to use the AWS Management Console to perform tasks.
- **AWS SDKs** – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access AWS Cloud Map. SDKs simplify authentication, integrate easily with your development environment, and provide access to AWS Cloud Map commands. For more information, see Tools for Amazon Web Services.
- **AWS Command Line Interface** – For more information, see Get started with the AWS CLI in the *AWS Command Line Interface User Guide*.
- **AWS Tools for Windows PowerShell** – For more information, see Get started with the AWS Tools for Windows PowerShell in the *AWS Tools for PowerShell User Guide*.
- **AWS Cloud Map API** – If you're using a programming language that an SDK isn't available for, see the AWS Cloud Map API Reference for information about API actions and about how to make API requests.

> **Note**
>
> **IPv6 Client Support** – As of June 22nd, 2023 in all new regions, any commands sent to AWS Cloud Map from IPv6 clients are routed to a new **dualstack endpoint** (`servicediscovery.<region>.api.aws`). AWS Cloud Map IPv6-only networks are reachable for both **legacy** (`servicediscovery.<region>.amazonaws.com`) and **dualstack endpoint**s in the following regions that were released prior to June 22nd, 2023:
>
> - US East (Ohio) – us-east-2

- US East (N. Virginia) – us-east-1
- US West (N. California) – us-west-1
- US West (Oregon) – us-west-2
- Africa (Cape Town) – af-south-1
- Asia Pacific (Hong Kong) – ap-east-1
- Asia Pacific (Hyderabad) – ap-south-2
- Asia Pacific (Jakarta) – ap-southeast-3
- Asia Pacific (Melbourne) – ap-southeast-4
- Asia Pacific (Mumbai) – ap-south-1
- Asia Pacific (Osaka) – ap-northeast-3
- Asia Pacific (Seoul) – ap-northeast-2
- Asia Pacific (Singapore) – ap-southeast-1
- Asia Pacific (Sydney) – ap-southeast-2
- Asia Pacific (Tokyo) – ap-northeast-1
- Canada (Central) – ca-central-1
- Europe (Frankfurt) – eu-central-1
- Europe (Ireland) – eu-west-1
- Europe (London) – eu-west-2
- Europe (Milan) – eu-south-1
- Europe (Paris) – eu-west-3
- Europe (Spain) – eu-south-2
- Europe (Stockholm) – eu-north-1
- Europe (Zurich) – eu-central-2
- Middle East (Bahrain) – me-south-1
- Middle East (UAE) – me-central-1
- South America (São Paulo) – sa-east-1
- AWS GovCloud (US-East) – us-gov-east-1
- AWS GovCloud (US-West) – us-gov-west-1

AWS Identity and Access Management

AWS Cloud Map integrates with AWS Identity and Access Management (IAM), a service that your organization can use to do the following actions:

- Create users and groups under your organization's AWS account
- Share your AWS account resources among the users in the account in an efficient manner
- Assign unique security credentials to each user
- Granularly control user access to services and resources

For example, you can use IAM with AWS Cloud Map to control which users in your AWS account can create a new namespace or register instances.

For general information about IAM, see the following resources:

- Identity and Access Management for AWS Cloud Map
- AWS Identity and Access Management
- IAM User Guide

AWS Cloud Map Pricing

AWS Cloud Map pricing is based on resources that you register in the service registry and API calls that you make to discover them. With AWS Cloud Map there are no upfront payments, and you only pay for what you use.

Optionally, you can enable DNS-based discovery for the resources with IP addresses. You can also enable health checking for your resources using Amazon Route 53 health checks, whether you're discovering instances using API calls or DNS queries. You will incur additional charges related to Route 53 DNS and health check usage.

For more information, see AWS Cloud Map Pricing.

AWS Cloud Map and AWS Cloud Compliance

For information about AWS Cloud Map compliance with various security compliance regulations and audits standards, see the following pages:

- AWS Cloud Compliance
- AWS Services in Scope by Compliance Program

Getting started with AWS Cloud Map

The following guides show you how to set up to use AWS Cloud Map and perform common tasks using AWS Cloud Map namespaces.

Guide overview	Learn more
Signing up for AWS and preparing to use AWS Cloud Map	Set up to use AWS Cloud Map
Using DNS queries and API calls to discover backend services.	Learn how to use AWS Cloud Map service discovery with DNS queries and API calls
Using DNS queries and API calls to discover backend services using the AWS CLI.	Learn how to use AWS Cloud Map service discovery with DNS queries and API calls using the AWS CLI
Creating a sample application and using custom attributes in code to discover resources.	Learn how to use AWS Cloud Map service discovery with custom attributes
Creating a sample application and using custom attributes in code to discover resources using the AWS CLI.	Learn how to use AWS Cloud Map service discovery with custom attributes using the AWS CLI

Set up to use AWS Cloud Map

The overview and procedures in the following sections are meant to help you get started with AWS and prepare you to start using AWS Cloud Map.

Topics

- Sign Up for AWS
- Access the API, AWS CLI, AWS Tools for Windows PowerShell, or the AWS SDKs
- Set Up the AWS Command Line Interface or AWS Tools for Windows PowerShell
- Download an AWS SDK

Sign Up for AWS

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Access the API, AWS CLI, AWS Tools for Windows PowerShell, or the AWS SDKs

To use the API, the AWS CLI, AWS Tools for Windows PowerShell, or the AWS SDKs, you must create *access keys*. These keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS.

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.

Which user needs programmatic access?	To	By
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Set Up the AWS Command Line Interface or AWS Tools for Windows PowerShell

The AWS Command Line Interface (AWS CLI) is a unified tool for managing AWS services. For information about how to install and configure the AWS CLI, see Installing or updating to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

If you have experience with Windows PowerShell, you might prefer to use AWS Tools for Windows PowerShell. For more information, see Setting up the AWS Tools for Windows PowerShell in the *AWS Tools for PowerShell User Guide*.

Download an AWS SDK

If you're using a programming language that AWS provides an SDK for, we recommend that you use an SDK instead of the AWS Cloud Map API. Using an SDK has several benefits. SDKs make authentication simpler, integrate easily with your development environment, and provide access to AWS Cloud Map commands. For more information, see Tools for Amazon Web Services.

Learn how to use AWS Cloud Map service discovery with DNS queries and API calls

The following tutorial simulates a microservice architecture with two backend services. The first service will be discoverable using a DNS query. The second service will be discoverable using the AWS Cloud Map API only.

> **Note**
>
> The resource details, like domain names and IP addresses, are for simulation purposes only. They can't be resolved over the internet.

For an end-to-end AWS CLI version of this tutorial, see Learn how to use AWS Cloud Map service discovery with DNS queries and API calls using the AWS CLI.

Prerequisites

The following prerequisites must be met to complete the tutorial successfully.

- Before you begin, complete the steps in Set up to use AWS Cloud Map.
- If you have not yet installed the AWS Command Line Interface, follow the steps at Installing or updating the latest version of the AWS CLI to install it.

  The tutorial requires a command line terminal or shell to run commands. In Linux and macOS, use your preferred shell and package manager.

> **Note**
>
> In Windows, some Bash CLI commands that you commonly use with Lambda (such as `zip`) are not supported by the operating system's built-in terminals. To get a Windows-integrated version of Ubuntu and Bash, install the Windows Subsystem for Linux.

- The tutorial requires a local environment with the `dig` DNS lookup utility command.

Step 1: Create an AWS Cloud Map namespace

In this step, you create a public AWS Cloud Map namespace. AWS Cloud Map creates a Route 53 hosted zone on your behalf with this same name. This gives you the ability to discover the service instances created in this namespace either using public DNS records or by using AWS Cloud Map API calls.

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. Choose **Create namespace**.
3. For **Namespace name**, specify `cloudmap-tutorial.com`.

   > **Note**
   >
   > If you were going to use this in production, you'd want to ensure that you specified the name of a domain you owned or had access to. But for the purposes of this tutorial, it's not necessary for it to be an actual domain that's being used.

4. (Optional) For **Namespace description**, specify a description for what you intend to use the namespace for.
5. For **Instance discovery**, select **API calls and public DNS queries**.
6. Leave the rest of the default values and choose **Create namespace**.

Step 2: Create the AWS Cloud Map services

In this step, you create two services. The first service will be discoverable using public DNS and API calls. The second service will be discoverable using API calls only.

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the left navigation pane, choose **Namespaces** to list the namespaces you've created.
3. From the list of namespaces, select the `cloudmap-tutorial.com` namespace and choose **View details**.
4. In the **Services** section, choose **Create service** and do the following to create the first service.

   a. For **Service name**, enter `public-service`. The service name will be applied to the DNS records that AWS Cloud Map creates. The format that is used is *<service-name>*.*<namespace-name>*.

   b. For **Service Discovery Configuration**, select **API and DNS**.

   c. In the **DNS configuration** section, for **Routing policy**, select **Multivalue answer routing**.

   > **Note**
   >
   > The console will translate this to **MULTIVALUE** after it is selected. For more information about available routing options, see Choosing a routing policy in the *Route 53 Developer Guide*.

   d. Leave the rest of the default values and choose **Create service** which will return you to the namespace details page.
5. In the **Services** section, choose **Create service** and do the following to create the second service.

   a. For **Service name**, enter `backend-service`.

   b. For **Service Discovery Configuration**, select **API only**.

   c. Leave the rest of the default values and choose **Create service**.

Step 3: Register the AWS Cloud Map service instances

In this step, you create two service instances, one for each service in our namespace.

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. From the list of namespaces, select the namespace you created in step 1 and choose **View details**.

3. On the namespace details page, from the list of services, select the `public-service` service and choose **View details**.
4. In the **Service instances** section, choose **Register service instance** and do the following to create the first service instance.
   a. For **Service instance ID**, specify `first`.
   b. For **IPv4 address**, specify `192.168.2.1`.
   c. Leave the rest of the default values and choose **Register service instance**.
5. Using the breadcrumb at the top of the page, select **cloudmap-tutorial.com** to navigate back to the namespace detail page.
6. On the namespace details page, from the list of services, select the **backend-service** service and choose **View details**.
7. In the **Service instances** section, choose **Register service instance** and do the following to create the second service instance.
   a. For **Service instance ID**, specify `second` to indicate that this is the second service instance.
   b. For **Instance type**, select **Identifying information for another resource**.
   c. For **Custom attributes**, add a key-value pair with `service-name` as the key and `backend` as the value.
   d. Choose **Register service instance**.

Step 4: Discover the AWS Cloud Map service instances

Now that the AWS Cloud Map namespace, services, and service instances are created, you can verify everything is working by discovering the instances. Use the `dig` command to verify the public DNS settings and the AWS Cloud Map API to verify the backend service. For more information about the `dig` command, see dig - DNS lookup utility.

1. Sign in to the AWS Management Console and open the Route 53 console at https://console.aws.amazon.com/route53/.
2. In the left navigation, choose **Hosted zones**.
3. Select the **cloudmap-tutorial.com** hosted zone. This displays the hosted zone details in a separate pane. Take note of the **Name servers** associated with your hosted zone as we will use those in the next step.

4. Using the dig command and one of the Route 53 name servers for your hosted zone, query the DNS records for your service instance.

```
dig @hosted-zone-nameserver public-service.cloudmap-tutorial.com
```

   The `ANSWER SECTION` in the output should display the IPv4 address you associated with your `public-service` service.

```
;; ANSWER SECTION:
public-service.cloudmap-tutorial.com. 300 IN A 192.168.2.1
```

5. Using the AWS CLI, query the attributes for your second service instances.

```
aws servicediscovery discover-instances --namespace-name cloudmap-tutorial.com --
service-name backend-service --region region
```

   The output displays the attributes you associated with the service as key-value pairs.

```
{
    "Instances": [
        {
            "InstanceId": "second",
            "NamespaceName": "cloudmap-tutorial.com",
            "ServiceName": "backend-service",
            "HealthStatus": "UNKNOWN",
            "Attributes": {
                "service-name": "backend"
            }
        }
    ],
    "InstancesRevision": 71462688285136850
}
```

Step 5: Clean up the resources

Once you have completed the tutorial, you can delete the resources. AWS Cloud Map requires that you clean them up in reverse order, the service instances first, then the services, and finally the namespace. AWS Cloud Map will clean up the Route 53 resources on your behalf when you go through these steps.

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. From the list of namespaces, select the `cloudmap-tutorial.com` namespace and choose **View details**.
3. On the namespace details page, from the list of services, select the `public-service` service and choose **View details**.
4. In the **Service instances** section, select the `first` instance and choose **Deregister**.
5. Using the breadcrumb at the top of the page, select **cloudmap-tutorial.com** to navigate back to the namespace detail page.
6. On the namespace details page, from the list of services, select the **public-service** service and choose **Delete**.
7. Repeat steps 3-6 for the `backend-service`.
8. In the left navigation, choose **Namespaces**.
9. Select the `cloudmap-tutorial.com` namespace and choose **Delete**.

> **Note**
>
> Although AWS Cloud Map cleans up the Route 53 resources on your behalf, you can navigate to the Route 53 console to verify that the `cloudmap-tutorial.com` hosted zone is deleted.

Learn how to use AWS Cloud Map service discovery with DNS queries and API calls using the AWS CLI

This tutorial demonstrates how to use AWS Cloud Map service discovery using the AWS Command Line Interface (CLI). You'll create a microservice architecture with two backend services – one discoverable using DNS queries and another discoverable using the AWS Cloud Map API only.

For a tutorial that includes AWS Cloud Map console steps, see Learn how to use AWS Cloud Map service discovery with DNS queries and API calls.

Prerequisites

The following prerequisites must be met to complete the tutorial successfully.

- Before you begin, complete the steps in Set up to use AWS Cloud Map.
- If you have not yet installed the AWS Command Line Interface, follow the steps at Installing or updating the latest version of the AWS CLI to install it.

  The tutorial requires a command line terminal or shell to run commands. In Linux and macOS, use your preferred shell and package manager.

  > **Note**
  >
  > In Windows, some Bash CLI commands that you commonly use with Lambda (such as `zip`) are not supported by the operating system's built-in terminals. To get a Windows-integrated version of Ubuntu and Bash, install the Windows Subsystem for Linux.

- The tutorial requires a local environment with the `dig` DNS lookup utility command.

Create an AWS Cloud Map namespace

First, you'll create a public AWS Cloud Map namespace. AWS Cloud Map will create a Route 53 hosted zone with the same name, enabling service discovery through both DNS records and API calls.

1. Create the public DNS namespace:

   ```
   aws servicediscovery create-public-dns-namespace \
       --name cloudmap-tutorial.com \
       --creator-request-id cloudmap-tutorial-request-1 \
       --region us-east-2
   ```

   The command returns an operation ID that you can use to check the status of the namespace creation:

   ```
   {
       "OperationId": "gv4g5meo7ndmeh4fqskygvk23d2fijwa-k9xmplyzd"
   }
   ```

2. Check the operation status to confirm the namespace was created successfully:

   ```
   aws servicediscovery get-operation \
       --operation-id gv4g5meo7ndmeh4fqskygvk23d2fijwa-k9xmplyzd \
   ```

```
    --region us-east-2
```

3. Once the operation is successful, get the namespace ID:

```
aws servicediscovery list-namespaces \
    --region us-east-2 \
    --query "Namespaces[?Name=='cloudmap-tutorial.com'].Id" \
    --output text
```

This command returns the namespace ID, which you'll need for subsequent steps:

```
ns-abcd1234xmplefgh
```

Create the AWS Cloud Map services

Now, create two services within your namespace. The first service will be discoverable using both DNS and API calls, while the second will be discoverable using API calls only.

1. Create the first service with DNS discovery enabled:

```
aws servicediscovery create-service \
    --name public-service \
    --namespace-id ns-abcd1234xmplefgh \
    --dns-config "RoutingPolicy=MULTIVALUE,DnsRecords=[{Type=A,TTL=300}]" \
    --region us-east-2
```

The command returns details about the created service:

```
{
    "Service": {
        "Id": "srv-abcd1234xmplefgh",
        "Arn": "arn:aws:servicediscovery:us-east-2:123456789012:service/srv-
abcd1234xmplefgh",
        "Name": "public-service",
        "NamespaceId": "ns-abcd1234xmplefgh",
        "DnsConfig": {
            "NamespaceId": "ns-abcd1234xmplefgh",
            "RoutingPolicy": "MULTIVALUE",
            "DnsRecords": [
                {
```

```
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "CreateDate": 1673613600.000,
        "CreatorRequestId": "public-service-request"
    }
}
```

2. Create the second service with API-only discovery:

```
aws servicediscovery create-service \
    --name backend-service \
    --namespace-id ns-abcd1234xmplefgh \
    --type HTTP \
    --region us-east-2
```

The command returns details about the created service:

```
{
    "Service": {
        "Id": "srv-ijkl5678xmplmnop",
        "Arn": "arn:aws:servicediscovery:us-east-2:123456789012:service/srv-
ijkl5678xmplmnop",
        "Name": "backend-service",
        "NamespaceId": "ns-abcd1234xmplefgh",
        "Type": "HTTP",
        "CreateDate": 1673613600.000,
        "CreatorRequestId": "backend-service-request"
    }
}
```

Register the AWS Cloud Map service instances

Next, register service instances for each of your services. These instances represent the actual resources that will be discovered.

1. Register the first instance with an IPv4 address for DNS discovery:

```
aws servicediscovery register-instance \
```

```
    --service-id srv-abcd1234xmplefgh \
    --instance-id first \
    --attributes AWS_INSTANCE_IPV4=192.168.2.1 \
    --region us-east-2
```

The command returns an operation ID:

```
{
    "OperationId": "4yejorelbukcjzpnr6tlmrghsjwpngf4-k9xmplyzd"
}
```

2. Check the operation status to confirm the instance was registered successfully:

```
aws servicediscovery get-operation \
    --operation-id 4yejorelbukcjzpnr6tlmrghsjwpngf4-k9xmplyzd \
    --region us-east-2
```

3. Register the second instance with custom attributes for API discovery:

```
aws servicediscovery register-instance \
    --service-id srv-ijkl5678xmplmnop \
    --instance-id second \
    --attributes service-name=backend \
    --region us-east-2
```

The command returns an operation ID:

```
{
    "OperationId": "7zxcvbnmasdfghjklqwertyuiop1234-k9xmplyzd"
}
```

4. Check the operation status to confirm the instance was registered successfully:

```
aws servicediscovery get-operation \
    --operation-id 7zxcvbnmasdfghjklqwertyuiop1234-k9xmplyzd \
    --region us-east-2
```

Discover the AWS Cloud Map service instances

Now that you've created and registered your service instances, you can verify everything is working by discovering them using both DNS queries and the AWS Cloud Map API.

1. First, get the Route 53 hosted zone ID:

```
aws route53 list-hosted-zones-by-name \
    --dns-name cloudmap-tutorial.com \
    --query "HostedZones[0].Id" \
    --output text
```

   This returns the hosted zone ID:

```
/hostedzone/Z1234ABCDXMPLEFGH
```

2. Get the name servers for your hosted zone:

```
aws route53 get-hosted-zone \
    --id Z1234ABCDXMPLEFGH \
    --query "DelegationSet.NameServers[0]" \
    --output text
```

   This returns one of the name servers:

```
ns-1234.awsdns-12.org
```

3. Use the `dig` command to query the DNS records for your public service:

```
dig @ns-1234.awsdns-12.org public-service.cloudmap-tutorial.com
```

   The output should display the IPv4 address you associated with your service:

```
;; ANSWER SECTION:
public-service.cloudmap-tutorial.com. 300 IN A 192.168.2.1
```

4. Use the AWS CLI to discover the backend service instance:

```
aws servicediscovery discover-instances \
    --namespace-name cloudmap-tutorial.com \
```

```
    --service-name backend-service \
    --region us-east-2
```

The output displays the attributes you associated with the service:

```
{
    "Instances": [
        {
            "InstanceId": "second",
            "NamespaceName": "cloudmap-tutorial.com",
            "ServiceName": "backend-service",
            "HealthStatus": "UNKNOWN",
            "Attributes": {
                "service-name": "backend"
            }
        }
    ],
    "InstancesRevision": 71462688285136850
}
```

Clean up the resources

Once you've completed the tutorial, clean up the resources to avoid incurring charges. AWS Cloud Map requires that you clean them up in reverse order: service instances first, then services, and finally the namespace.

1. Deregister the first service instance:

```
aws servicediscovery deregister-instance \
    --service-id srv-abcd1234xmplefgh \
    --instance-id first \
    --region us-east-2
```

2. Deregister the second service instance:

```
aws servicediscovery deregister-instance \
    --service-id srv-ijkl5678xmplmnop \
    --instance-id second \
    --region us-east-2
```

3. Delete the public service:

```
aws servicediscovery delete-service \
    --id srv-abcd1234xmplefgh \
    --region us-east-2
```

4. Delete the backend service:

```
aws servicediscovery delete-service \
    --id srv-ijkl5678xmplmnop \
    --region us-east-2
```

5. Delete the namespace:

```
aws servicediscovery delete-namespace \
    --id ns-abcd1234xmplefgh \
    --region us-east-2
```

6. Verify that the Route 53 hosted zone is deleted:

```
aws route53 list-hosted-zones-by-name \
    --dns-name cloudmap-tutorial.com
```

Learn how to use AWS Cloud Map service discovery with custom attributes

The following tutorial demonstrates how you can use AWS Cloud Map service discovery with custom attributes that are discoverable using the AWS Cloud Map API. The tutorial walks you through creating and running client applications using AWS CloudShell. The applications use two Lambda functions to write data to a DynamoDB table and then read from the table. The Lambda functions and DynamoDB table are registered in AWS Cloud Map as service instances. The code in the client applications and Lambda functions uses AWS Cloud Map custom attributes to discover the resources needed to perform the job.

For an AWS CLI-based version of this tutorial, see Learn how to use AWS Cloud Map service discovery with custom attributes using the AWS CLI.

> **Important**
>
> You will create AWS resources during the workshop which will incur a cost in your AWS account. It is recommended to clean-up the resources as soon as you finish the workshop to minimize the cost.

Prerequisites

Before you begin, complete the steps in Set up to use AWS Cloud Map.

Step 1: Create an AWS Cloud Map namespace

In this step, you create an AWS Cloud Map namespace. A namespace is a construct used to group services for an application. When you create the namespace, you specify how the resources will be discoverable. The resources created in the namespace created in this step will be discoverable with AWS Cloud Map API calls using custom attributes.

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. Choose **Create namespace**.
3. For **Namespace name**, specify `cloudmap-tutorial`.
4. (Optional) For **Namespace description**, specify a description for what you intend to use the namespace for.
5. For **Instance discovery**, select **API calls**.
6. Leave the rest of the default values and choose **Create namespace**.

Step 2: Create a DynamoDB table

In this step, you create a DynamoDB table. The table is used to store and retrieve data for the sample application that you will create in the following steps.

For information about how to create an DynamoDB, see Step 1: Create a table in DynamoDB in the *DynamoDB Developer Guide* and use the following table to determine what options to specify.

Option	Value
Table name	cloudmap
Partition key	id

Keep the default values for the rest of the settings and create the table.

Step 3: Create an AWS Cloud Map data service and register DynamoDB table as an instance

In this step, you create a AWS Cloud Map service and then register the DynamoDB table created in the last step as a service instance.

1. Open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/
2. From the list of namespaces, select the `cloudmap-tutorial` namespace and choose **View details**.
3. In the **Services** section, choose **Create service** and do the following.
   a. For **Service name**, enter `data-service`.
   b. Leave the rest of the default values and choose **Create service**.
4. In the **Services** section, select the `data-service` service and choose **View details**.
5. In the **Service instances** section, choose **Register service instance**.
6. On the **Register service instance** page, do the following.
   a. For **Instance type**, select **Identifying information for another resource**.
   b. For **Service instance id**, specify `data-instance`.
   c. In the **Custom attributes** section, specify the following key-value pair: **key** = `tablename`, **value** = `cloudmap`.

Step 4: Create an AWS Lambda execution role

In this step, you create an IAM role that the AWS Lambda function in the next step uses. You can name the IAM role `cloudmap-tutorial-role` and omit the permissions boundary because the role is only used for this tutorial, and you can delete it afterwards.

To create the service role for Lambda (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Lambda**, and then choose the **Lambda** use case.
5. Choose **Next**.
6. Search for, and select the box next to, the `PowerUserAccess` policy and then choose **Next**.
7. Choose **Next**.
8. For **Role name**, specify `cloudmap-tutorial-role`.
9. Review the role, and then choose **Create role**.

Step 5: Create the Lambda function to write data

In this step, you create a Lambda function authored from scratch that writes data to the DynamoDB table by using the AWS Cloud Map API to query the AWS Cloud Map service you created.

For information about creating a Lambda function, see Create a Lambda function with the console in the *AWS Lambda Developer Guide* and use the following table to determine what options to specify or choose.

Option	Value
Function name	writefunction
Runtime	Python 3.12
Architecture	x86_64
Permissions	Use an existing role
Existing role	cloudmap-tutorial-role

After you create the function, update the example code to reflect the following Python code, and then deploy the function. Note that you're specifying the `datatable` custom attribute you associated with the AWS Cloud Map service instance you created for the DynamoDB table. The function generates a key that is a random number between 1 and 100 and associates it with a value that is passed to the function when it is called.

```
import json
import boto3
import random

def lambda_handler(event, context):

    serviceclient = boto3.client('servicediscovery')

    response = serviceclient.discover_instances(
        NamespaceName='cloudmap-tutorial',
        ServiceName='data-service')

    tablename = response["Instances"][0]["Attributes"]["tablename"]

    dynamodbclient = boto3.resource('dynamodb')

    table = dynamodbclient.Table(tablename)

    response = table.put_item(
        Item={ 'id': str(random.randint(1,100)), 'todo': event })

    return {
        'statusCode': 200,
        'body': json.dumps(response)
    }
```

After deploying the function, to avoid timeout errors, update the function timeout to 5 seconds. For more information, see Configure Lambda function timeout in the *AWS Lambda Developer Guide*.

Step 6: Create an AWS Cloud Map app service and register the Lambda write function as an instance

In this step, you create an AWS Cloud Map service and then register the Lambda write function as a service instance.

1. Open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/
2. In the left navigation, choose **Namespaces**.
3. From the list of namespaces, select the `cloudmap-tutorial` namespace and choose **View details**.
4. In the **Services** section, choose **Create service** and do the following.
   a. For **Service name**, enter `app-service`.
   b. Leave the rest of the default values and choose **Create service**.
5. In the **Services** section, select the `app-service` service and choose **View details**.
6. In the **Service instances** section, choose **Register service instance**.
7. On the **Register service instance** page, do the following.
   a. For **Instance type**, select **Identifying information for another resource**.
   b. For **Service instance id**, specify `write-instance`.
   c. In the **Custom attributes** section, specify the following key-value pairs.
      - **key** = `action`, **value** = `write`
      - **key** = `functionname`, **value** = `writefunction`

Step 7: Create the Lambda function to read data

In this step, you create a Lambda function authored from scratch that writes data to the DynamoDB table you created.

For information about creating a Lambda function, see Create a Lambda function with the console in the *AWS Lambda Developer Guide* and use the following table to determine what options to specify or choose.

Option	Value
Function name	readfunction
Runtime	Python 3.12
Architecture	x86_64

Option	Value
Permissions	Use an existing role
Existing role	cloudmap-tutorial-role

After you create the function, update the example code to reflect the following Python code, and then deploy the function. The function scans the table amd returns all items.

```
import json
import boto3

def lambda_handler(event, context):
    serviceclient = boto3.client('servicediscovery')

    response = serviceclient.discover_instances(NamespaceName='cloudmap-tutorial',
 ServiceName='data-service')

    tablename = response["Instances"][0]["Attributes"]["tablename"]

    dynamodbclient = boto3.resource('dynamodb')

    table = dynamodbclient.Table(tablename)

    response = table.scan(Select='ALL_ATTRIBUTES')

    return {
        'statusCode': 200,
        'body': json.dumps(response)
    }
```

After deploying the function, to avoid timeout errors, update the function timeout to 5 seconds. For more information, see Configure Lambda function timeout in the *AWS Lambda Developer Guide*.

Step 8: Register the Lambda read function as an AWS Cloud Map service instance

In this step, you register the Lambda read function as a service instance in the `app-service` service you previously created.

1. Open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/
2. In the left navigation, choose **Namespaces**.
3. From the list of namespaces, select the `cloudmap-tutorial` namespace and choose **View details**.
4. In the **Services** section, select the `app-service` service and choose **View details**.
5. In the **Service instances** section, choose **Register service instance**.
6. On the **Register service instance** page, do the following.

   a. For **Instance type**, select **Identifying information for another resource**.

   b. For **Service instance id**, specify `read-instance`.

   c. In the **Custom attributes** section, specify the following key-value pairs.

   - **key** = `action`, **value** = `read`
   - **key** = `functionname`, **value** = `readfunction`

Step 9: Create and run read and write clients on AWS CloudShell

You can create and run client applications in AWS CloudShell that use code to discover the services you configured in AWS Cloud Map and make calls to these services.

1. Open the AWS CloudShell console at https://console.aws.amazon.com/cloudshell/
2. Use the following command to create a file called `writefunction.py`.

```
vim writeclient.py
```

3. In the `writeclient.py` file, enter insert mode by pressing the `i` button. Then, copy and paste the following code. This code discovers the Lambda function to write data by searching for the custom attribute `name=writeservice` in the `app-service` service. The name of the Lambda function responsible for writing data to the DynamoDB table is returned. Then the Lambda function is invoked, passing a sample payload that is written to the table as a value.

```
import boto3

serviceclient = boto3.client('servicediscovery')

response = serviceclient.discover_instances(NamespaceName='cloudmap-tutorial',
 ServiceName='app-service', QueryParameters={ 'action': 'write' })
```

```
functionname = response["Instances"][0]["Attributes"]["functionname"]

lambdaclient = boto3.client('lambda')

resp = lambdaclient.invoke(FunctionName=functionname, Payload='"This is a test
 data"')

print(resp["Payload"].read())
```

4. Press the escape key, type `:wq`, and press the enter key to save the file and exit.
5. Use the following command to run the Python code.

```
python3 writeclient.py
```

The output should be a `200` response, similar to the following.

```
b'{"statusCode": 200, "body": "{\\"ResponseMetadata\\": {\\"RequestId\\": \
\"Q0M038IT0BPBVBJK8OCKK6I6M7VV4KQNSO5AEMVJF66Q9ASUAAJG\\", \\"HTTPStatusCode\
\": 200, \\"HTTPHeaders\\": {\\"server\\": \\"Server\\", \\"date\\": \\"Wed, 06
 Mar 2024 22:46:09 GMT\\", \\"content-type\\": \\"application/x-amz-json-1.0\\",
 \\"content-length\\": \\"2\\", \\"connection\\": \\"keep-alive\\", \\"x-amzn-
requestid\\": \\"Q0M038IT0BPBVBJK8OCKK6I6M7VV4KQNSO5AEMVJF66Q9ASUAAJG\\", \\"x-amz-
crc32\\": \\"2745614147\\"}, \\"RetryAttempts\\": 0}}"}'
```

6. To verify the write was successful in the previous step, create a read client.

   a. Use the following command to create a file called `readfunction.py`.

   ```
   vim readclient.py
   ```

   b. In the `readclient.py` file, press the `i` button to enter insert mode. Then, copy and paste the following code. This code scans the table and will return the value that you wrote to the table in the previous step.

   ```
   import boto3

   serviceclient = boto3.client('servicediscovery')

   response = serviceclient.discover_instances(NamespaceName='cloudmap-tutorial',
    ServiceName='app-service', QueryParameters={ 'action': 'read' })
   ```

```
functionname = response["Instances"][0]["Attributes"]["functionname"]

lambdaclient = boto3.client('lambda')

resp = lambdaclient.invoke(FunctionName=functionname,
 InvocationType='RequestResponse')

print(resp["Payload"].read())
```

c. Press the escape key, type `:wq`, and press the enter key to save the file and exit.

d. Use the following command to run the Python code.

```
python3 readclient.py
```

The output should look similar to the following, listing the value written to the table by running `writefunction.py` and the random key generated in the Lambda write function.

```
  b'{"statusCode": 200, "body": "{\\"Items\\": [{\\"id\\": \\"45\
\", \\"todo\\": \\"This is a test data\\"}], \\"Count\\": 1, \
\"ScannedCount\\": 1, \\"ResponseMetadata\\": {\\"RequestId\\": \
\"9JF8J6SFQCKR6IDT5JG5NOM3CNVV4KQNSO5AEMVJF66Q9ASUAAJG\\", \\"HTTPStatusCode\
\": 200, \\"HTTPHeaders\\": {\\"server\\": \\"Server\\", \\"date\\": \\"Thu, 25
 Jul 2024 20:43:33 GMT\\", \\"content-type\\": \\"application/x-amz-json-1.0\
\", \\"content-length\\": \\"91\\", \\"connection\\": \\"keep-alive\\", \\"x-
amzn-requestid\\": \\"9JF8J6SFQCKR6IDT5JG5NOM3CNVV4KQNSO5AEMVJF66Q9ASUAAJG\\",
 \\"x-amz-crc32\\": \\"1163081893\\"}, \\"RetryAttempts\\": 0}}"}'
```

Step 10: Clean up the resources

After you have completed the tutorial, delete the resources to avoid incurring additional charges. AWS Cloud Map requires that you clean them up in reverse order, the service instances first, then the services, and finally the namespace. The following steps walk you through cleaning up the AWS Cloud Map resources used in the tutorial.

To delete the AWS Cloud Map resources

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.

2. From the list of namespaces, select the `cloudmap-tutorial` namespace and choose **View details**.
3. On the namespace details page, from the list of services, select the `data-service` service and choose **View details**.
4. In the **Service instances** section, select the `data-instance` instance and choose **Deregister**.
5. Using the breadcrumb at the top of the page, select **cloudmap-tutorial.com** to navigate back to the namespace detail page.
6. On the namespace details page, from the list of services, select the **data-service** service and choose **Delete**.
7. Repeat steps 3-6 for the `app-service` service and the `write-instance` and `read-instance` service instances.
8. In the left navigation, choose **Namespaces**.
9. Select the `cloudmap-tutorial` namespace and choose **Delete**.

The following table lists procedures that you can follow to delete the other resources used in the tutorial.

Resource	Steps
DynamoDB table	Step 6: (Optional) Delete your DynamoDB table to clean up resources in the *Amazon DynamoDB Developer Guide*
Lambda functions and associated IAM execution role	Clean up in the *AWS Lambda Developer Guide*

Learn how to use AWS Cloud Map service discovery with custom attributes using the AWS CLI

This tutorial demonstrates how you can use AWS Cloud Map service discovery with custom attributes. You'll create a microservices application that uses AWS Cloud Map to discover resources dynamically using custom attributes. The application consists of two Lambda functions that write data to and read from a DynamoDB table, with all resources registered in AWS Cloud Map.

For an AWS Management Console version of the tutorial, see Learn how to use AWS Cloud Map service discovery with custom attributes.

Prerequisites

Before you begin this tutorial, complete the steps in Set up to use AWS Cloud Map.

Create an AWS Cloud Map namespace

A namespace is a construct used to group services for an application. In this step, you'll create a namespace that allows resources to be discoverable through AWS Cloud Map API calls.

1. Run the following command to create an HTTP namespace:

   ```
   aws servicediscovery create-http-namespace \
     --name cloudmap-tutorial \
     --creator-request-id cloudmap-tutorial-request
   ```

   The command returns an operation ID. You can check the status of the operation with the following command:

   ```
   aws servicediscovery get-operation \
     --operation-id operation-id
   ```

2. Once the namespace is created, you can retrieve its ID for use in subsequent commands:

   ```
   aws servicediscovery list-namespaces \
     --query "Namespaces[?Name=='cloudmap-tutorial'].Id" \
     --output text
   ```

3. Store the namespace ID in a variable for later use:

   ```
   NAMESPACE_ID=$(aws servicediscovery list-namespaces \
     --query "Namespaces[?Name=='cloudmap-tutorial'].Id" \
     --output text)
   ```

Create a DynamoDB table

Next, create a DynamoDB table that will store data for your application:

1. Run the following command to create the table:

```
aws dynamodb create-table \
  --table-name cloudmap \
  --attribute-definitions AttributeName=id,AttributeType=S \
  --key-schema AttributeName=id,KeyType=HASH \
  --billing-mode PAY_PER_REQUEST
```

2. Wait for the table to become active before proceeding:

```
aws dynamodb wait table-exists --table-name cloudmap
```

   This command waits until the table is fully created and ready to use.

Create an AWS Cloud Map data service and register the DynamoDB table

Now, create a service in your namespace to represent data storage resources:

1. Run the following command to create a AWS Cloud Map service for data storage resources:

```
aws servicediscovery create-service \
  --name data-service \
  --namespace-id $NAMESPACE_ID \
  --creator-request-id data-service-request
```

2. Get the service ID for the data service:

```
DATA_SERVICE_ID=$(aws servicediscovery list-services \
  --query "Services[?Name=='data-service'].Id" \
  --output text)
```

3. Register the DynamoDB table as a service instance with a custom attribute that specifies the table name:

```
aws servicediscovery register-instance \
  --service-id $DATA_SERVICE_ID \
  --instance-id data-instance \
  --attributes tablename=cloudmap
```

The custom attribute `tablename=cloudmap` allows other services to discover the DynamoDB table name dynamically.

Create an IAM role for Lambda functions

Create an IAM role that the Lambda functions will use to access AWS resources:

1. Create the trust policy document for the IAM role using the following JSON:

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "lambda.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. Run the following command to create the IAM role using the trust policy:

   ```
   aws iam create-role \
     --role-name cloudmap-tutorial-role \
     --assume-role-policy-document file://lambda-trust-policy.json
   ```

3. Create a file for a custom IAM policy with least privilege permissions using the following JSON:

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
   ```

```
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "servicediscovery:DiscoverInstances"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/cloudmap"
    }
  ]
}
```

4. Create and attach the policy to the IAM role:

```
aws iam create-policy \
  --policy-name CloudMapTutorialPolicy \
  --policy-document file://cloudmap-policy.json

POLICY_ARN=$(aws iam list-policies \
  --query "Policies[?PolicyName=='CloudMapTutorialPolicy'].Arn" \
  --output text)

aws iam attach-role-policy \
  --role-name cloudmap-tutorial-role \
  --policy-arn $POLICY_ARN

aws iam attach-role-policy \
  --role-name cloudmap-tutorial-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Create the Lambda function to write data

To create a Lambda function that writes data to the DynamoDB table, follow these steps:

1. Create the Python file for the write function:

```
cat > writefunction.py << EOF
import json
import boto3
import random

def lambda_handler(event, context):
    try:
        serviceclient = boto3.client('servicediscovery')

        response = serviceclient.discover_instances(
            NamespaceName='cloudmap-tutorial',
            ServiceName='data-service')

        if not response.get("Instances"):
            return {
                'statusCode': 500,
                'body': json.dumps({"error": "No instances found"})
            }

        tablename = response["Instances"][0]["Attributes"].get("tablename")
        if not tablename:
            return {
                'statusCode': 500,
                'body': json.dumps({"error": "Table name attribute not found"})
            }

        dynamodbclient = boto3.resource('dynamodb')

        table = dynamodbclient.Table(tablename)

        # Validate input
        if not isinstance(event, str):
            return {
                'statusCode': 400,
                'body': json.dumps({"error": "Input must be a string"})
            }
```

```
        response = table.put_item(
            Item={ 'id': str(random.randint(1,100)), 'todo': event })

        return {
            'statusCode': 200,
            'body': json.dumps(response)
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({"error": str(e)})
        }
EOF
```

This function uses AWS Cloud Map to discover the DynamoDB table name from the custom attribute, then writes data to the table.

2. Package and deploy the Lambda function:

```
zip writefunction.zip writefunction.py

ROLE_ARN=$(aws iam get-role --role-name cloudmap-tutorial-role \
  --query 'Role.Arn' --output text)

aws lambda create-function \
  --function-name writefunction \
  --runtime python3.12 \
  --role $ROLE_ARN \
  --handler writefunction.lambda_handler \
  --zip-file fileb://writefunction.zip \
  --architectures x86_64
```

3. Update the function timeout to avoid timeout errors:

```
aws lambda update-function-configuration \
  --function-name writefunction \
  --timeout 5
```

Create an AWS Cloud Map app service and register the Lambda write function

To create another service in your namespace to represent application functions, follow these steps:

1. Create a service for application functions:

```
aws servicediscovery create-service \
  --name app-service \
  --namespace-id $NAMESPACE_ID \
  --creator-request-id app-service-request
```

2. Get the service ID for the app service:

```
APP_SERVICE_ID=$(aws servicediscovery list-services \
  --query "Services[?Name=='app-service'].Id" \
  --output text)
```

3. Register the Lambda write function as a service instance with custom attributes:

```
aws servicediscovery register-instance \
  --service-id $APP_SERVICE_ID \
  --instance-id write-instance \
  --attributes action=write,functionname=writefunction
```

   The custom attributes `action=write` and `functionname=writefunction` allow clients to discover this function based on its purpose.

Create the Lambda function to read data

To create a Lambda function that reads data from the DynamoDB table, follow these steps:

1. Create the Python file for the read function:

```
cat > readfunction.py << EOF
import json
import boto3

def lambda_handler(event, context):
    try:
```

```
        serviceclient = boto3.client('servicediscovery')

        response = serviceclient.discover_instances(
            NamespaceName='cloudmap-tutorial',
            ServiceName='data-service')

        if not response.get("Instances"):
            return {
                'statusCode': 500,
                'body': json.dumps({"error": "No instances found"})
            }

        tablename = response["Instances"][0]["Attributes"].get("tablename")
        if not tablename:
            return {
                'statusCode': 500,
                'body': json.dumps({"error": "Table name attribute not found"})
            }

        dynamodbclient = boto3.resource('dynamodb')

        table = dynamodbclient.Table(tablename)

        # Use pagination for larger tables
        response = table.scan(
            Select='ALL_ATTRIBUTES',
            Limit=50  # Limit results for demonstration purposes
        )

        # For production, you would implement pagination like this:
        # items = []
        # while 'LastEvaluatedKey' in response:
        #     items.extend(response['Items'])
        #     response = table.scan(
        #         Select='ALL_ATTRIBUTES',
        #         ExclusiveStartKey=response['LastEvaluatedKey']
        #     )
        # items.extend(response['Items'])

        return {
            'statusCode': 200,
            'body': json.dumps(response)
        }
    except Exception as e:
```

```
        return {
            'statusCode': 500,
            'body': json.dumps({"error": str(e)})
        }
EOF
```

This function also uses AWS Cloud Map to discover the DynamoDB table name, then reads data from the table. It includes error handling and pagination comments.

2. Package and deploy the Lambda function:

```
zip readfunction.zip readfunction.py

aws lambda create-function \
  --function-name readfunction \
  --runtime python3.12 \
  --role $ROLE_ARN \
  --handler readfunction.lambda_handler \
  --zip-file fileb://readfunction.zip \
  --architectures x86_64
```

3. Update the function timeout:

```
aws lambda update-function-configuration \
  --function-name readfunction \
  --timeout 5
```

Register the Lambda read function as a service instance

To register the Lambda read function as another service instance in the app service, follow this step:

```
aws servicediscovery register-instance \
  --service-id $APP_SERVICE_ID \
  --instance-id read-instance \
  --attributes action=read,functionname=readfunction
```

The custom attributes `action=read` and `functionname=readfunction` allow clients to discover this function based on its purpose.

Create and run client applications

To create a Python client application that uses AWS Cloud Map to discover and invoke the write function, follow these steps:

1. Create a Python file for the write client application:

```
cat > writeclient.py << EOF
import boto3
import json

try:
    serviceclient = boto3.client('servicediscovery')

    print("Discovering write function...")
    response = serviceclient.discover_instances(
        NamespaceName='cloudmap-tutorial',
        ServiceName='app-service',
        QueryParameters={ 'action': 'write' }
    )

    if not response.get("Instances"):
        print("Error: No instances found")
        exit(1)

    functionname = response["Instances"][0]["Attributes"].get("functionname")
    if not functionname:
        print("Error: Function name attribute not found")
        exit(1)

    print(f"Found function: {functionname}")

    lambdaclient = boto3.client('lambda')

    print("Invoking Lambda function...")
    resp = lambdaclient.invoke(
        FunctionName=functionname,
        Payload='"This is a test data"'
    )

    payload = resp["Payload"].read()
    print(f"Response: {payload.decode('utf-8')}")
```

```
except Exception as e:
    print(f"Error: {str(e)}")
EOF
```

This client uses the `QueryParameters` option to find service instances with the `action=write` attribute.

2. Create a Python file for the read client application:

```
cat > readclient.py << EOF
import boto3
import json

try:
    serviceclient = boto3.client('servicediscovery')

    print("Discovering read function...")
    response = serviceclient.discover_instances(
        NamespaceName='cloudmap-tutorial',
        ServiceName='app-service',
        QueryParameters={ 'action': 'read' }
    )

    if not response.get("Instances"):
        print("Error: No instances found")
        exit(1)

    functionname = response["Instances"][0]["Attributes"].get("functionname")
    if not functionname:
        print("Error: Function name attribute not found")
        exit(1)

    print(f"Found function: {functionname}")

    lambdaclient = boto3.client('lambda')

    print("Invoking Lambda function...")
    resp = lambdaclient.invoke(
        FunctionName=functionname,
        InvocationType='RequestResponse'
    )

    payload = resp["Payload"].read()
```

```
    print(f"Response: {payload.decode('utf-8')}")

except Exception as e:
    print(f"Error: {str(e)}")
EOF
```

3. Run the write client to add data to the DynamoDB table:

```
python3 writeclient.py
```

The output should show a successful response with HTTP status code 200.

4. Run the read client to retrieve data from the DynamoDB table:

```
python3 readclient.py
```

The output should show the data that was written to the table, including the randomly generated ID and the "This is a test data" value.

Clean up resources

When you're finished with the tutorial, clean up the resources to avoid incurring additional charges.

1. First, run the following command to deregister the service instances:

```
aws servicediscovery deregister-instance \
  --service-id $APP_SERVICE_ID \
  --instance-id read-instance

aws servicediscovery deregister-instance \
  --service-id $APP_SERVICE_ID \
  --instance-id write-instance

aws servicediscovery deregister-instance \
  --service-id $DATA_SERVICE_ID \
  --instance-id data-instance
```

2. Run the following command to delete the services:

```
aws servicediscovery delete-service \
  --id $APP_SERVICE_ID
```

```
aws servicediscovery delete-service \
  --id $DATA_SERVICE_ID
```

3. Run the following command to delete the namespace:

```
aws servicediscovery delete-namespace \
  --id $NAMESPACE_ID
```

4. Run the following command to delete the Lambda functions:

```
aws lambda delete-function --function-name writefunction
aws lambda delete-function --function-name readfunction
```

5. Run the following command to delete the IAM role and policy:

```
aws iam detach-role-policy \
  --role-name cloudmap-tutorial-role \
  --policy-arn $POLICY_ARN

aws iam detach-role-policy \
  --role-name cloudmap-tutorial-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

aws iam delete-policy \
  --policy-arn $POLICY_ARN

aws iam delete-role --role-name cloudmap-tutorial-role
```

6. Run the following command to delete the DynamoDB table:

```
aws dynamodb delete-table --table-name cloudmap
```

7. Run the following command to clean up temporary files:

```
rm -f lambda-trust-policy.json cloudmap-policy.json writefunction.py
 readfunction.py writefunction.zip readfunction.zip writeclient.py readclient.py
```

AWS Cloud Map namespaces

A namespace is a logical entity in AWS Cloud Map that is used to group an application's services under a common name and level of discoverability. When you create a namespace, you specify the following:

- A name that you want your application to use to discover instances.
- The method by which service instances that you register with AWS Cloud Map can be discovered. You can decide whether your resources need to be discovered publicly over the internet, privately in a specific virtual private cloud (VPC), or by API calls only.

The following are general concepts about namespaces.

- Namespaces are specific to the AWS Region that they're created in. To use AWS Cloud Map in multiple regions, you'll need to create namespaces in each region.
- If you create a namespace to allow for instance discovery by DNS queries in a VPC, AWS Cloud Map automatically creates a private Route 53 hosted zone. This hosted zone can be associated with multiple VPCs. For more information, see AssociateVPCWithHostedZone in the *Amazon Route 53 API Reference*.

Topics

Creating an AWS Cloud Map namespace to group application services

You can create a namespace to group services for your application under a friendly name that allows for the discovery of application resources through API calls or DNS queries.

Instance discovery options

The following table summarizes the different instance discovery options in AWS Cloud Map and the corresponding namespace type that you can create, depending on your application's services and setup.

Namespace type	Instance discovery method	How it works	Additional information
HTTP	API calls	Resources in your application can discover other resources by calling the `DiscoverInstances` API only.	• DiscoverInstances • CreateHttpNamespace
Private DNS	API calls and DNS queries in a VPC	When you create a private DNS namespace, AWS Cloud Map creates a corresponding Amazon Route 53 private hosted zone. Resources in your application can discover other resources by calling the `DiscoverInstances` API, and by querying the nameservers in the private Route 53 hosted zone that AWS Cloud Map automatically creates.	• DiscoverInstances • CreatePrivateDnsNamespace

Namespace type	Instance discovery method	How it works	Additional information
		The hosted zone created by AWS Cloud Map has the same name as the namespace and contains DNS records that have names in the format *service-name* .*namespace-name*. **Note** Route 53 Resolver resolves DNS queries that originate in the VPC using records in the private hosted zone. If the private hosted zone doesn't include a record that matches the domain name in a DNS query, Route 53 responds to the	

Namespace type	Instance discovery method	How it works	Additional information
		query with NXDOMAIN (non-existent domain).	

Namespace type	Instance discovery method	How it works	Additional information
Public DNS	API calls and public DNS queries	When you create a public DNS namespace, AWS Cloud Map creates a corresponding Amazon Route 53 public hosted zone. Resources in your application can discover other resources by calling the `DiscoverInstances` API and by querying the nameservers in the public Route 53 hosted zone that AWS Cloud Map automatically creates. The public hosted zone has the same name as the namespace and contains DNS records that have names in the format *`service-name`*`.`*`namespace-name`*. **Note** The namespace	• DiscoverInstances • CreatePublicDnsNamespace

Namespace type	Instance discovery method	How it works	Additional information
		name in this case must be a domain name that you've registered.	

Procedure

You can follow these steps to create a namespace using the AWS CLI, AWS Management Console, or the SDK for Python.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. Choose **Create namespace**.
3. For **Namespace name**, enter a name that will be used to discover instances.

> **Note**
>
> - Namespaces configured for public DNS queries must end with a top level domain. For example, `.com`.
> - You can specify an internationalized domain name (IDN) if you convert the name to Punycode first. For information about online converters, perform an internet search on "punycode converter".
>
> You can also convert an internationalized domain name to Punycode when you create namespaces programmatically. For example, if you're using Java, you can convert a Unicode value to Punycode by using the `toASCII` method of the java.net.IDN library.

4. (Optional) For **Namespace description**, enter information about the namespace that will be visible on the **Namespaces** page and under **Namespace information**. You can use this information to easily identify a namespace.
5. For **Instance discovery**, you can choose between **API calls**, **API calls and DNS queries in VPCs**, and **API calls and public DNS queries** to create a HTTP, private DNS, or public DNS namespace respectively. For more information, see Instance discovery options.

   Based on your selection, follow these steps.

   - If you choose **API calls and DNS queries in VPCs**, for **VPC**, choose a virtual private cloud (VPC) that you want to associate the namespace with.
   - If you choose **API calls and DNS queries in VPCs** or **API calls and public DNS queries**, for **TTL**, specify a numerical value in seconds. The time to live (TTL) value determines how long DNS resolvers cache information for the start of authority (SOA) DNS record of the Route 53 hosted zone created with your namespace. For more information about TTL, see TTL (seconds) in the *Amazon Route 53 Developer Guide*.
6. (Optional) Under **Tags**, choose **Add tags** and then specify a key and a value to tag your namespace. You can specify one or more tags to add to your namespace. Tags allow you to categorize your AWS resources so you can more easily manage them. For more information, see Tagging your AWS Cloud Map resources.
7. Choose **Create namespace**. You can view the status of the operation by using ListOperations. For more information, see ListOperations in the *AWS Cloud Map API Reference*

AWS CLI

- Create a namespace with the command for the instance discovery type you would prefer (replace the *red* values with your own).
  - Create an HTTP namespace using `create-http-namespace`. Service instances registered using an HTTP namespace can be discovered using a `DiscoverInstances` request, but they can't be discovered using DNS.

    ```
    aws servicediscovery create-http-namespace --name name-of-namespace
    ```

  - Create a private namespace based on DNS and only visible inside a specified Amazon VPC using `create-private-dns-namespace`. You can discover instances that were

registered with a private DNS namespace by using either a `DiscoverInstances` request or using DNS

```
aws servicediscovery create-private-dns-namespace --name name-of-namespace --
vpc vpc-xxxxxxxxx
```

- Create a public namespace based on DNS that is visible on the internet using `create-public-dns-namespace`. You can discover instances that were registered with a public DNS namespace by using either a `DiscoverInstances` request or using DNS.

```
aws servicediscovery create-public-dns-namespace --name name-of-namespace
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. Create a namespace with the command for the instance discovery type you would prefer (replace the *red* values with your own):

   - Create an HTTP namespace using `create_http_namespace()`. Service instances registered using an HTTP namespace can be discovered using `discover_instances()`, but they can't be discovered using DNS.

```
response = client.create_http_namespace(
    Name='name-of-namespace',
)
# If you want to see the response
print(response)
```

   - Create a private namespace based on DNS and only visible inside a specified Amazon VPC using `create_private_dns_namespace()`. You can discover instances that were registered with a private DNS namespace by using either `discover_instances()` or using DNS

```
response = client.create_private_dns_namespace(
    Name='name-of-namespace',
    Vpc='vpc-1c56417b',
)
# If you want to see the response
print(response)
```

- Create a public namespace based on DNS that is visible on the internet using `create_public_dns_namespace()`. You can discover instances that were registered with a public DNS namespace by using either `discover_instances()` or using DNS.

```
response = client.create_public_dns_namespace(
    Name='name-of-namespace',
)
# If you want to see the response
print(response)
```

- Example response output

```
{
    'OperationId': 'gv4g5meo7ndmeh4fqskygvk23d2fijwa-k9302yzd',
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Next steps

After creating a namespace, you can create services in the namespace to group together application resources that collectively serve a particular purpose in your application. A service acts as a template for registering application resources as instances. For more information about creating AWS Cloud Map services, see Creating an AWS Cloud Map service for an application component.

Listing AWS Cloud Map namespaces

After creating namespaces, you can view a list of the namespaces you've created by following these steps.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces** to view a list of namespaces. You can order namespaces by name, description, instance discovery mode, owner, or namespace ID. You can also enter a namespace name or ID into the search field to locate and view a specific namespace.

AWS CLI

- List namespaces with the `list-namespaces` command.

```
aws servicediscovery list-namespaces
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. List namespaces with `list_namespaces()`.

```
response = client.list_namespaces()
# If you want to see the response
print(response)
```

Example response output

```
{
    'Namespaces': [
        {
            'Arn': 'arn:aws::servicediscovery:us-west-2:123456789012:namespace/
ns-xxxxxxxxxxxxxx',
            'CreateDate': 1585354387.357,
```

```
            'Id': 'ns-xxxxxxxxxxxxxx',
            'Name': 'myFirstNamespace',
            'Properties': {
                'DnsProperties': {
                    'HostedZoneId': 'Z06752353VBUDTC32S84S',
                },
                'HttpProperties': {
                    'HttpName': 'myFirstNamespace',
                },
            },
            'Type': 'DNS_PRIVATE',
        },
        {
            'Arn': 'arn:aws::servicediscovery:us-west-2:123456789012:namespace/
ns-xxxxxxxxxxxxxx',
            'CreateDate': 1586468974.698,
            'Description': 'My second namespace',
            'Id': 'ns-xxxxxxxxxxxxxx',
            'Name': 'mySecondNamespace.com',
            'Properties': {
                'DnsProperties': {
                },
                'HttpProperties': {
                    'HttpName': 'mySecondNamespace.com',
                },
            },
            'Type': 'HTTP',
        },
        {
            'Arn': 'arn:aws::servicediscovery:us-west-2:123456789012:namespace/
ns-xxxxxxxxxxxxxx',
            'CreateDate': 1587055896.798,
            'Id': 'ns-xxxxxxxxxxxxxx',
            'Name': 'myThirdNamespace.com',
            'Properties': {
                'DnsProperties': {
                    'HostedZoneId': 'Z09983722P0QME1B3KC8I',
                },
                'HttpProperties': {
                    'HttpName': 'myThirdNamespace.com',
                },
            },
            'Type': 'DNS_PRIVATE',
        },
```

```
    ],
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Deleting an AWS Cloud Map namespace

After you're done using a namespace, you can delete it. When you delete a namespace, you can no longer use it to register or discover service instances.

> **Note**
>
> When you delete a DNS namespace, AWS Cloud Map deletes the corresponding Amazon Route 53 hosted zone created during namespace creation.

Before deleting a namespace, you must deregister all service instances and then delete all services that were created in the namespace. For more information, see Deregistering an AWS Cloud Map service instance and Deleting an AWS Cloud Map service.

After you've deregistered instances and deleted services that were created in a namespace, follow these steps to delete the namespace.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. Select the namespace that you want to delete, then choose **Delete**.
4. Confirm that you want to delete the service by choosing **Delete** again.

AWS CLI

- Delete a namespace with the `delete-namespace` command (replace the *red* value with your own). If the namespace still contains one or more services, the request fails.

```
aws servicediscovery delete-namespace --id ns-xxxxxxxxxxx
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. Delete a namespace with `delete_namespace()` (replace the *red* value with your own). If the namespace still contains one or more services, the request fails.

```
response = client.delete_namespace(
    Id='ns-xxxxxxxxxxx',
)
# If you want to see the response
print(response)
```

Example response output

```
{
    'OperationId': 'gv4g5meo7ndmeh4fqskygvk23d2fijwa-k98y6drk',
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Shared AWS Cloud Map namespaces

AWS Cloud Map allows namespace owners to share their namespaces with other AWS accounts or within an organization in AWS Organizations for simplified cross-account service discovery and service registry. This allows for easier use of namespaces managed by other AWS accounts or teams within an AWS Organization.

AWS Cloud Map integrates with AWS Resource Access Manager (AWS RAM) to enable resource sharing. AWS RAM is a service that enables you to share some AWS Cloud Map resources with other AWS accounts or through AWS Organizations. With AWS RAM, you share resources that you own by creating a *resource share*. A resource share specifies the resources to share, and the consumers with whom to share them. Consumers can include:

- Specific AWS accounts inside its organization in AWS Organizations
- An organizational unit inside its organization in AWS Organizations
- Its entire organization in AWS Organizations

For more information about AWS RAM, see the *AWS RAM User Guide*.

This topic explains how to share resources that you own, and how to use resources that are shared with you.

Contents

Considerations for sharing namespaces

- To share a namespace, you must own it in your AWS account. This means that the resource must be allocated or provisioned in your account. You can't share a namespace that has been shared with you.
- To share a namespace with your organization or an organizational unit in AWS Organizations, you must enable sharing with AWS Organizations. For more information, see Enable Sharing with AWS Organizations in the *AWS RAM User Guide*.

- For service discovery using DNS queries in a shared private DNS namespace, the namespace owner will need to call `create-vpc-association-authorization` with the ID of the private hosted zone associated with the namespace and the consumer's VPC.

  ```
  aws route53 create-vpc-association-authorization --hosted-zone-id Z1234567890ABC --
  vpc VPCRegion=us-east-1,VPCId=vpc-12345678
  ```

  The namespace consumer will need to call `associate-vpc-with-hosted-zone` with the ID of the private hosted zone.

  ```
  aws route53 associate-vpc-with-hosted-zone --hosted-zone-id Z1234567890ABC --vpc
   VPCRegion=us-east-1,VPCId=vpc-12345678
  ```

  For more information, see Associating an Amazon VPC and a private hosted zone that you created with different AWS accounts in the *Amazon Route 53 Developer Guide*.
- After discovering up-to-date network locations of services associated with a shared DNS namespace, it may be necessary to configure inter-VPC connectivity to communicate with the services if they are in different VPCs. This can be achieved using a VPC Peering connection. For more information, see Create or delete a VPC Peering connection in the *Amazon Virtual Private Cloud VPC Peering guide*.
- You can't use `ListOperations` to list operations on shared namespaces that are performed by other accounts.
- Tagging isn't supported for shared namespaces.

Sharing an AWS Cloud Map namespace

When you share an AWS Cloud Map namespace that you own with other AWS accounts (consumers), you enable these accounts to discover the up-to-date network locations of services in the namespace without the need for temporary credentials.

To share a namespace, you must add it to a resource share. A resource share is an AWS RAM resource that lets you share your resources across AWS accounts. A resource share specifies the resources to share, and the consumers with whom they are shared. To add the namespace to a new resource share, you must first create the resource share using the AWS RAM console.

If you are part of an organization in AWS Organizations and sharing within your organization is enabled, consumers in your organization are automatically granted access to the shared

namespace. Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared namespace after accepting the invitation.

You can share a namespace that you own using the AWS RAM console or the AWS CLI.

AWS RAM console

To share a namespace that you own using the AWS RAM console

See Creating a resource share in AWS RAM in the *AWS RAM User Guide*.

AWS CLI

To share a namespace that you own using the AWS CLI

Use the AWS RAM create-resource-share command.

Stop sharing a AWS Cloud Map namespace

When a namespace is no longer shared, the namespace and any services and instances associated with it can no longer be accessed by consumer AWS accounts. This includes resources created in the namespace by consumers when they had access to the namespace.

To stop sharing a namespace that you own, you must remove it from the resource share. You can do this using the AWS RAM console or the AWS CLI.

AWS RAM console

To stop sharing a namespace that you own using the AWS RAM console

See Updating a Resource Share in the *AWS RAM User Guide*.

AWS CLI

To stop sharing a namespace that you own using the AWS CLI

Use the disassociate-resource-share command.

Identifying a shared AWS Cloud Map namespace

Owners and consumers can identify shared namespaces using the AWS Cloud Map console and AWS CLI. The namespace owner can be identified by using the `ResourceOwner` property. The AWS

account that creates a service or registers an instance in the shared namespace can be identified by using the `CreatedByAccount` property.

AWS Cloud Map console

To identify a shared namespace using the AWS Cloud Map console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. On the **Namespaces** page, under **Resource Owner**, you can find the ID of the AWS account that owns the namespace.
3. Choose the **Domain name** of the namespace you want to identify.
4. On the **Namespace: *namespace-name*** page, in the **Namespace information** section, under **Resource owner**, you can find the ID of the AWS account that owns the namespace.

AWS CLI

To identify a shared namespace using the AWS CLI, use the list-namespaces command. The command returns the namespaces that you own and namespaces that are shared with you. The `ResourceOwner` field shows the AWS account ID of the namespace owner.

The following `list-namespaces` call is made by account `111122223333`.

```
aws servicediscovery list-namespaces
```

Output:

```
{
    "Namespaces": [
        {
            "Arn": "arn:aws:servicediscovery:us-west-2:111122223333:namespace/ns-
abcdef01234567890",
            "CreateDate": 1585354387.357,
            "Id": "ns-abcdef01234567890",
            "Name": "local",
            "Properties": {
                "DnsProperties": {
                    "HostedZoneId": "Z06752353VBUDTC32S84S"
                },
```

```
                "HttpProperties": {
                    "HttpName": "local"
                 }
            },
            "Type": "DNS_PRIVATE",
            "ServiceCount": 2,
           "ResourceOwner": "111122223333"
        },
        {
            "Arn": "arn:aws:servicediscovery:us-west-2:444455556666:namespace/
ns-021345abcdef6789",
            "CreateDate": 1586468974.698,
            "Description": "Shared second namespace",
            "Id": "ns-021345abcdef6789",
            "Name": "My-second-namespace",
            "Properties": {
                "DnsProperties": {},
                "HttpProperties": {
                    "HttpName": "Shared-second-namespace"
                }
            },
            "Type": "HTTP",
            "ServiceCount": 0,
            "ResourceOwner": "444455556666"
        }
    ]
}
```

In this scenario, namespace `ns-abcdef01234567890` is created and owned by `111122223333` and namespace `ns-021345abcdef6789` is created and owned by `444455556666`. Namespace `ns-021345abcdef6789` is shared with account `111122223333` by account `444455556666`.

Granting permissions to share a namespace

A minimum set of permissions is required for an IAM principal to share a namespace. We recommend using the `AWSCloudMapFullAccess` and `AWSResourceAccessManagerFullAccess` managed policies to ensure your IAM principals have the required permissions to share and use shared namespaces.

If you use a custom IAM policy, the `servicediscovery:PutResourcePolicy`, `servicediscovery:GetResourcePolicy`, and

`servicediscovery:DeleteResourcePolicy` actions are required for sharing namespaces. These are permission-only IAM actions. If an IAM principal doesn't have these permissions granted, an error will occur when attempting to share the namespace using AWS RAM.

For more information about how AWS RAM uses IAM, see How AWS RAM uses IAM in the *AWS RAM User Guide*.

Responsibilities and permissions for shared namespaces

The namespace owner and consumer can perform different actions on a shared namespace.

Permissions for owners

A namespace owner can perform the following actions on a shared namespace:

- Access services associated with the namespace, including services created by consumer accounts and instances registered to these services.
- Revoke access to the namespace, including access to services created by consumer accounts and instances registered to these services.
- Configure permissions for other accounts to register and deregister instances in services created in the shared namespace by consumers or the namespace owner.
- Delete services and deregister instances, including services created and instances registered by consumer accounts.
- Update or delete a shared namespace.

Permissions for consumers

A namespace consumer can perform the following actions on a shared namespace:

- Create and delete services in the namespace.
- Register and deregister instances in services created in the namespace.
- Discover instances that are registered to services created in the namespace.

A consumer can't update or delete a shared namespace. After losing access to the shared namespace, the consumer accounts will also lose access to services that they created in the namespace.

Billing and metering

Owners are billed for any instances that they register in the shared namespace and any Route 53 health checks that are created when they register these instances. Consumers are billed for any instances that they register in the namespace and any Route 53 health checks that are created when they register these instances. If the shared namespace is a DNS namespace, the namespace owner is billed for the Route 53 DNS records that are created when services are created in the namespace. Owners are billed for any `DiscoverInstances` and `DiscoverInstancesRevision` calls they make. Consumers are billed for any `DiscoverInstances` and `DiscoverInstancesRevision` calls they make.

Quotas

Shared namespaces count towards only the namespace owner's namespaces per Region quota. Instances registered by a consumer in the shared namespace count towards the owner's instances per namespace quota. If a consumer creates a service in a shared namespace, any instances registered in the service count towards the consumer's instances per service quota. If an owner creates a service in a shared namespace, any instances registered in the service count towards the owner's instances per service quota.

AWS Cloud Map services

An AWS Cloud Map service is a template for registering service instances that consists of the service name and DNS configuration, if applicable, for the service. You can also set up a health check to determine the health status of instances in the service and filter out unhealthy resources. A service can represent a component of your application. For example, you can create a service for resources that handle payments on your application and another for resources that manage users.

A service allows you to locate the resources for an application by getting back one or more endpoints that can be used to connect to the resource. The location of resources is done using DNS queries or the AWS Cloud Map `DiscoverInstances` API action, depending on how you've configured the namespace. You can use the AWS Cloud Map console to scope instance discovery at the service level.

You can also specify custom metadata as atttributes at the service level using the `UpdateServiceAttributes` API. You can set service attributes to avoid duplicating attributes across instances and modify these attributes without needing to make any changes to instance attributes. Information you can specify as attributes at the service level includes, but isn't limited to, the following:

- Endpoint weights for shifting traffic during progressive deployments.
- Service preferences such as API timeouts and suggested retry policies.

For more information, see UpdateServiceAttributes in the *AWS Cloud Map API reference*.

The following topics describe health check and DNS configurations for services and include instructions for creating, listing, updating, and deleting a service.

Topics

AWS Cloud Map service health check configuration

Health checks help determine whether service instances are healthy or not. If you don't configure a health check during service creation, traffic will be routed to service instances regardless of the instances' health status. When you configure a health check, AWS Cloud Map returns healthy resources by default. You can use the `HealthStatus` parameter of the `DiscoverInstances` API to filter resources by health status and get a list of unhealthy resources. You can also use the `GetInstancesHealthStatus` API to retrieve the health status of a particular service instance.

You can either configure a Route 53 health check or a custom, third-party health check when you create an AWS Cloud Map service.

Route 53 health checks

If you specify settings for an Amazon Route 53 health check, AWS Cloud Map creates a Route 53 health check whenever you register an instance and deletes the health check when you deregister the instance.

For public DNS namespaces, AWS Cloud Map associates the health check with the Route 53 record that AWS Cloud Map creates when you register an instance.If you specify both `A` and `AAAA` record types in a service's DNS configuration, AWS Cloud Map creates a health check that uses the IPv4 address to check the health of the resource. If the endpoint that's specified by the IPv4 address is unhealthy, Route 53 considers both the `A` and `AAAA` records to be unhealthy. If you specify a `CNAME` record type in a service's DNS configuration, you can't configure a Route 53 health check.

For namespaces that you use API calls to discover instances for, AWS Cloud Map creates a Route 53 health check. However, there's no DNS record for AWS Cloud Map to associate the health check with. To determine whether a health check is healthy, you can configure monitoring using either the Route 53 console or using Amazon CloudWatch. For more information about using the Route 53 console, see Get Notified When a Health Check Fails in the *Amazon Route 53 Developer Guide*. For more information about using CloudWatch, see PutMetricAlarm in the *Amazon CloudWatch API Reference*.

Note

- You can't configure an Amazon Route 53 health check for a service created in a private DNS namespace.

- A Route 53 health checker in each health-checking AWS Region sends a health check request to an endpoint every 30 seconds. On average, your endpoint receives a health check request about every two seconds. However, health checkers don't coordinate with one another. Therefore, you might sometimes see several requests in one second that's followed by a few seconds with no health checks at all. For a list of health-checking regions, see Regions.

For information about the charges for Route 53 health checks, see Route 53 Pricing.

Custom health checks

If you configure AWS Cloud Map to use a custom health check when you register an instance, you must use a third-party health checker to evaluate the health of your resources. Custom health checks are useful in the following circumstances:

- You can't use a Route 53 health check because the resource isn't available over the internet. For example, suppose that you have an instance that's located in an Amazon VPC. You can use a custom health check for this instance. However, for the health check to work,your health checker must also be in the same VPC as your instance.
- You want to use a third-party health checker regardless of where your resources are.

When you use a custom health checks, AWS Cloud Map doesn't check the health of a given resource directly. Instead, the third-party health checker checks the health of the resource and returns a status to your application. Your application will then need to submit a `UpdateInstanceCustomHealthStatus` request that relays this status to AWS Cloud Map. If the initial status relayed is `UNHEALTHY`, and if there isn't another `UpdateInstanceCustomHealthStatus` within 30 seconds that relays a status of `HEALTHY`, the resource is confirmed to be unhealthy. AWS Cloud Map stops routing traffic to that resource.

AWS Cloud Map service DNS configuration

When you create a service in a namespace that supports instance discovery by DNS queries, AWS Cloud Map creates Route 53 DNS records. You must specify a Route 53 routing policy and DNS record type that will apply to all Route 53 DNS records that AWS Cloud Map creates.

Routing policy

A routing policy determines how Route 53 responds to the DNS queries that are used for service instance discovery. Supported routing policies and how they relate to AWS Cloud Map are as follows.

Weighted routing

Route 53 returns the applicable value from one randomly selected AWS Cloud Map service instance from among the instances that you registered using the same AWS Cloud Map service. All records have the same weight, so you can't route more or less traffic to any instances.

For example, suppose the service includes configurations for one **A** record and a health check, and you use the service to register 10 instances. Route 53 responds to DNS queries with the IP address for one randomly selected instance from among the healthy instances. If no instances are healthy, Route 53 responds to DNS queries as if all the instances were healthy.

If you don't define a health check for the service, Route 53 assumes that all instances are healthy and returns the applicable value for one randomly selected instance.

For more information, see Weighted Routing in the *Amazon Route 53 Developer Guide*.

Multivalue answer routing

If you define a health check for the service and the result of the health check is healthy, Route 53 returns the applicable value for up to eight instances.

For example, suppose that the service includes configurations for one **A** record and a health check. You use the service to register 10 instances. Route 53 responds to DNS queries with IP addresses for only a maximum of eight healthy instances. If fewer than eight instances are healthy, Route 53 responds to every DNS query with the IP addresses for all the healthy instances.

If you don't define a health check for the service, Route 53 assumes that all instances are healthy and returns the values for up to eight instances.

For more information, see Multivalue Answer Routing in the *Amazon Route 53 Developer Guide*.

Record type

A Route 53 DNS record type determines the type of value Route 53 returns in response to the DNS queries that are used for service instance discovery. The different DNS record types you can specify, and the associated values returned by Route 53 in response to queries are as follows.

A

If you specify this type, Route 53 returns the IP address of the resource in IPv4 format, such as **192.0.2.44**.

AAAA

If you specify this type, Route 53 returns the IP address of the resource in IPv6 format, such as **2001:0db8:85a3:0000:0000:abcd:0001:2345**.

CNAME

If you specify this type, Route 53 returns the domain name of the resource (such as www.example.com).

> **Note**
>
> - To configure a **CNAME** DNS record, you must specify the **Weighted routing** routing policy.
> - When you configure a **CNAME** DNS record, you can't configure a Route 53 health check.

SRV

If you specify this type, Route 53 returns the value for an SRV record. The value for an **SRV** record uses the following values:

```
priority weight port service-hostname
```

Consider the following:

- The values of `priority` and `weight` are both set to 1 and can't be changed.
- For `port`, AWS Cloud Map uses the value that you specify for **Port** (AWS_INSTANCE_PORT) when you register an instance.

- The value of `service-hostname` is a concatenation of the following values:
  - The value that you specify for **Service instance ID** (InstanceID) when you register an instance
  - The name of the service
  - The name of the namespace

  For example, suppose you specify **test** as an instance ID when you register an instance. The name of the service is **backend** and the name of the namespace is **example.com**. AWS Cloud Map assigns the following value to the `service-hostname` attribute in the **SRV** record:

  `test.backend.example.com`

> **Note**
>
> If you specify values an IPv4 address, an IPv6 address, or both when you register an instance, AWS Cloud Map automatically creates **A** and/or **AAAA** records that have the same name as the value of `service-hostname` in the **SRV** record.

You can specify record types in the following combinations:

- **A**
- **AAAA**
- **A** and **AAAA**
- **CNAME**
- **SRV**

If you specify **A** and **AAAA** record types, you can specify an IPv4 IP address, an IPv6 IP address, or both when you register an instance.

Creating an AWS Cloud Map service for an application component

After creating a namespace, you can create services to represent different components of your application that serve particular purposes. For example, you can create a service for resources in your application that process payments.

> **Note**
>
> You can't create multiple services that are accessible by DNS queries with names that differ only by case (such as EXAMPLE and example). Trying to do so will result in these services having the same DNS name. If you use a namespace that's only accessible by API calls, then you can create services that with names that differ only by case.

Follow these steps to create a service using the AWS Management Console, AWS CLI, and SDK for Python.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. On the **Namespaces** page, choose the namespace that you want to add the service to.
4. On the **Namespace: *`namespace-name`*** page, choose **Create service**.
5. For **Service name**, enter a name that describes the instances that you register when using this service. The value is used to discover AWS Cloud Map service instances either in API calls or in DNS queries.

   > **Note**
   >
   > If you want AWS Cloud Map to create an **SRV** record when you register an instance and you're using a system that requires a specific **SRV** format (such as HAProxy), specify the following for **Service name**:
   >
   > - Start the name with an underscore (_), for example **_exampleservice**.
   > - End the name with *`._protocol`*, for example **._tcp**.
   >
   > When you register an instance, AWS Cloud Map creates an **SRV** record and assigns a name by concatenating the service name and the namespace name, for example: **_exampleservice._tcp.example.com**

6. (Optional) For **Service description**, enter a description for the service. The description that you enter here appears on the **Services** page and on the detail page for each service.

7. If the namespace supports DNS queries, under **Service discovery configuration**, you can configure discoverability at the service level. Choose between allowing both API calls and DNS queries or only API calls for the discovery of instances in this service.

   > **Note**
   >
   > If you choose **API calls**, AWS Cloud Map will not create SRV records when you register an instance.

   If you choose **API and DNS**, follow these steps to configure DNS records. You can add or remove DNS records.

   1. For **Routing policy**, select the Amazon Route 53 routing policy for the DNS records that AWS Cloud Map creates when you register instances. You can select between **Weighted routing** and **Multivalue answer routing**. For more information, see Routing policy.

      > **Note**
      >
      > You can't use the console to configure AWS Cloud Map to create a Route 53 alias record when you register an instance. If you want AWS Cloud Map to create alias records for an Elastic Load Balancing load balancer when you register instances programmatically, choose **Weighted routing** for **Routing policy**.

   2. For **Record type**, choose the DNS record type that determines what Route 53 returns in response to DNS queries by AWS Cloud Map. For more information, see Record type.
   3. For **TTL**, specify a numerical value to define the time to live (TTL) value, in seconds, at the service level. The value of TTL determines how long DNS resolvers cache information for this record before the resolvers forward another DNS query to Amazon Route 53 to get updated settings.

8. Under **Health check configuration**, for **Health check options**, choose the type of health check applicable for service instances. You can choose not to configure any health checks, or you can choose between a Route 53 health check or an external health check for your instances. For more information, see AWS Cloud Map service health check configuration.

> **Note**
>
> Route 53 health checks are configurable only for services in public DNS namespaces.

If you choose **Route 53 health checks**, provide the following information.

1. For **Failure threshold**, provide a number between 1 and 10 that defines the number of consecutive Route 53 health checks a service instance must pass or fail for its health status to change.
2. For **Health check protocol**, select the method Route 53 will use to check the health of the service instances.
3. If you choose **HTTP** or **HTTPS** health check protocol, for **Health check path**, provide a path that you want Amazon Route 53 to request when performing health checks. The path can be any value such as the file `/docs/route53-health-check.html`. When the resource is healthy, the returned value is an HTTP status code of a 2xx or 3xx format. You can also include query string parameters, for example, `/welcome.html?language=jp&login=y`. The AWS Cloud Map console automatically adds a leading slash (/) character.

For more information about Route 53 health checks, see How Amazon Route 53 Determines Whether a Health Check Is Healthy in the *Amazon Route 53 Developer Guide*.

9. (Optional) Under **Tags**, choose **Add tags** and then specify a key and a value to tag your namespace. You can specify one or more tags to add to your namespace. Tags allow you to categorize your AWS resources so you can more easily manage them. For more information, see Tagging your AWS Cloud Map resources.
10. Choose **Create service**.

AWS CLI

- Create a service with the `create-service` command. Replace the *red* values with your own.

```
aws servicediscovery create-service \
```

```
    --name service-name \
    --namespace-id  ns-xxxxxxxxxxx \
    --dns-config "NamespaceId=ns-
xxxxxxxxxxx,RoutingPolicy=MULTIVALUE,DnsRecords=[{Type=A,TTL=60}]"
```

Output:

```
{
        "Service": {
        "Id": "srv-xxxxxxxxxxx",
        "Arn": "arn:aws:servicediscovery:us-west-2:123456789012:service/srv-
xxxxxxxxxxx",
        "Name": "service-name",
        "NamespaceId": "ns-xxxxxxxxxxx",
        "DnsConfig": {
            "NamespaceId": "ns-xxxxxxxxxxx",
            "RoutingPolicy": "MULTIVALUE",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 60
                }
            ]
        },
        "CreateDate": 1587081768.334,
        "CreatorRequestId": "567c1193-6b00-4308-bd57-ad38a8822d25"
    }
}
```

AWS SDK for Python (Boto3)

If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.

1. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

2. Create a service with `create_service()`. Replace the *red* values with your own. For more information, see create_service.

```
response = client.create_service(
    DnsConfig={
        'DnsRecords': [
            {
                'TTL': 60,
                'Type': 'A',
            },
        ],
        'NamespaceId': 'ns-xxxxxxxxxxx',
        'RoutingPolicy': 'MULTIVALUE',
    },
    Name='service-name',
    NamespaceId='ns-xxxxxxxxxxx',
)
```

Example response output

```
{
    'Service': {
        'Arn': 'arn:aws:servicediscovery:us-west-2:123456789012:service/srv-
xxxxxxxxxxx',
        'CreateDate': 1587081768.334,
        'DnsConfig': {
            'DnsRecords': [
                {
                    'TTL': 60,
                    'Type': 'A',
                },
            ],
            'NamespaceId': 'ns-xxxxxxxxxxx',
            'RoutingPolicy': 'MULTIVALUE',
        },
        'Id': 'srv-xxxxxxxxxxx',
        'Name': 'service-name',
        'NamespaceId': 'ns-xxxxxxxxxxx',
    },
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Next steps

After creating a service, you can register your application resources as service instances that contain information about how your application can locate the resource. For more information about registering AWS Cloud Map service instances, see Registering a resource as an AWS Cloud Map service instance.

You can also specify custom metadata such as endpoint weights, API timeouts, and retry policies as service attributes after creating a service. For more information, see ServiceAttributes and `UpdateServiceAttributes` in the *AWS Cloud Map API Reference*.

Updating an AWS Cloud Map service

Depending on a service's configuration, you can update its tags, Route 53 health check failure threshold, and time to live (TTL) for DNS resolvers. To update a service, perform the following procedure.

> **Note**
>
> You can't update settings for services associated with HTTP namespaces.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. On the **Namespaces** page, choose the namespace in which the service is created.
4. On the **Namespace: *`namespace-name`*** page, select the service you want to edit and choose **View details**.
5. On the **Service: *`service-name`*** page, choose **Edit**.

   > **Note**
   >
   > You can't use the **Edit** button workflow to edit values for services that allow only API calls for instance discovery. However, you can add or remove tags on the **Service: *`service-name`*** page.

6. On the **Edit service** page, under **Service description**, you can update any previously set description for the service or add a new description. You can also add tags and update **TTL** for DNS resolvers.
7. Under **DNS configuration**, for **TTL**, you can specify an updated period of time, in seconds, that determines how long DNS resolvers cache information for this record before the resolvers forward another DNS query to Amazon Route 53 to get updated settings.
8. If you've set up Route 53 health checks, for **Failure threshold**, you can specify a new number between 1 and 10 that defines the number of consecutive Route 53 health checks a service instance must pass or fail for its health status to change.
9. Choose **Update service**.

AWS CLI

- Update a service with the `update-service` command (replace the *red* value with your own).

```
aws servicediscovery update-service \
    --id  srv-xxxxxxxxxxx \
    --service "Description=new
 description,DnsConfig={DnsRecords=[{Type=A,TTL=60}]}"
```

Output:

```
{
    "OperationId": "l3pfx7f4ynndrbj3cfq5fm2qy2z37bms-5m6iaoty"
}
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. Update a service with `update_service()` (replace the *red* value with your own).

```
response = client.update_service(
    Id='srv-xxxxxxxxxxx',
    Service={
        'DnsConfig': {
            'DnsRecords': [
                {
                    'TTL': 300,
                    'Type': 'A',
                },
            ],
        },
        'Description': "new description",
    }
)
```

Example response output

```
{
    "OperationId": "l3pfx7f4ynndrbj3cfq5fm2qy2z37bms-5m6iaoty"
}
```

Listing AWS Cloud Map services in a namespace

To view a list of the services that you created in a namespace, perform the following procedure.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. Choose the **Domain name** of the namespace that contains the services that you want to list. You can view a list of all services under **Services** and enter the service name or ID in the search field to find a specific service. You can identify the AWS account that created the service by using the **Created by** field and the account that owns the service by using the **Resource owner** field.

> **Note**
>
> If the namespace is a shared namespace, the AWS account ID under **Resource owner** is the account that created and shared the namespace. The account ID under **Created by** can differ from the ID under **Resource owner** if a namespace consumer created the service. The account IDs may not be the same as your account ID. For more information about shared namespaces, see Shared AWS Cloud Map namespaces.

AWS CLI

- List services with the `list-services` command. The following command lists all services in a namespace using the namespace ID as the filter. Replace the *red* value with your own.

```
aws servicediscovery list-services --filters
 Name=NAMESPACE_ID,Values=ns-1234567890abcdef,Condition=EQ
```

AWS SDK for Python (Boto3)

1. If you don't already have Boto3 installed, you can find instructions for installing, configuring, and using Boto3 here.
2. Import Boto3 and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. List services with `list_services()`.

```
response = client.list_services()
# If you want to see the response
print(response)
```

Example response output

```
{
    'Services': [
```

```
        {
            'Arn': 'arn:aws:servicediscovery:us-west-2:123456789012:service/srv-
xxxxxxxxxxxxxxx',
            'CreateDate': 1587081768.334,
            'DnsConfig': {
                'DnsRecords': [
                    {
                        'TTL': 60,
                        'Type': 'A',
                    },
                ],
                'RoutingPolicy': 'MULTIVALUE',
            },
            'Id': 'srv-xxxxxxxxxxxxxxxx',
            'Name': 'myservice',
        },
    ],
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Deleting an AWS Cloud Map service

Before you can delete a service, you must deregister all service instances that were registered using the service. For more information, see Deregistering an AWS Cloud Map service instance.

After deregistering all instances registered using the service, perform the following procedure to delete the service.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. Choose the option for the namespace that contains the service that you want to delete.
4. On the **Namespace: *namespace-name*** page, choose the option for the service that you want to delete.
5. Choose **Delete**.

6. Confirm that you want to delete the service.

AWS CLI

- Delete a service with the `delete-service` command (replace the *red* value with your own).

```
aws servicediscovery delete-service --id srv-xxxxxx
```

AWS SDK for Python (Boto3)

1. If you don't already have Boto3 installed, you can find instructions for installing, configuring, and using Boto3 here.
2. Import Boto3 and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. Delete a service with `delete_service()` (replace the *red* value with your own).

```
response = client.delete_service(
    Id='srv-xxxxxx',
)
# If you want to see the response
print(response)
```

Example response output

```
{
    'ResponseMetadata': {
        '...': '...',
    },
}
```

AWS Cloud Map service instances

A service instance contains information about how to locate a resource, such as a web server, for an application. After you register instances, you locate them by using DNS queries or the AWS Cloud Map DiscoverInstances API action. The resources you can register include, but aren't limited to, the following:

- Amazon EC2 instances
- Amazon DynamoDB tables
- Amazon S3 buckets
- Amazon Simple Queue Service (Amazon SQS) queues
- APIs deployed on top of Amazon API Gateway

You can specify attribute values for services instances, and clients can use these attributes to filter the resources that AWS Cloud Map returns. For example, an application can request resources in a particular deployment stage, like BETA or PROD. You can also use attributes for versioning.

The following procedures describe how you can register resources in your application as service instances, view a list of registered instances in a service, edit certain instance parameters, and deregister an instance.

Topics

Registering a resource as an AWS Cloud Map service instance

You can register your application's resources as instances in a AWS Cloud Map service. For example, assume you've created a service called `users` for all application resources that manage user data. You can then register a DynamoDB table that's used to store user data as an instance in this service.

Note

The following features are not available on the AWS Cloud Map console:

- When you register a service instance using the console, you can't create an alias record that routes traffic to an Elastic Load Balancing (ELB) load balancer. When you register an instance, you must include the `AWS_ALIAS_DNS_NAME` attribute. For more information, see RegisterInstance in the *AWS Cloud Map API Reference*.
- If you register an instance using a service that includes a custom health check, you can't specify the initial status for the custom health check. By default, the initial status of a custom health checks is **Healthy**. If you want the initial health status to be **Unhealthy**, register the instance programmatically and include the `AWS_INIT_HEALTH_STATUS` attribute. For more information, see RegisterInstance in the *AWS Cloud Map API Reference*.

To register an instance in a service, follow these steps.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. On the **Namespaces** page, choose the namespace that contains the service that you want to use as a template for registering a service instance.
4. On the **Namespace: *`namespace-name`*** page, choose the service that you want to use.
5. On the **Service: *`service-name`*** page, choose **Register service instance**.
6. On the **Register service instance** page, choose an **Instance type**. Depending on the namespace instance discovery configuration, you can choose to specify an IP address, an Amazon EC2 instance ID, or other identifying information for a resource that doesn't have an IP address.

   Note

   You can choose **EC2 instance** only in HTTP namespaces.

7. For **Service instance ID**, provide an identifier associated with the service instance.

> **Note**
>
> If you want to update an existing instance, provide the identifier associated with the instance you want to update. Then, use the next steps to update values and reregister the instance.

8. Based on your choice of **Instance type**, perform the following steps.

> **Important**
>
> You can't use the `AWS_` prefix (not case sensitive) in a key when you specify a custom attribute.

Instance type	Steps
IP address	a. Under **Standard attributes**, for **IPv4 address**, provide an IPv4 address, if any, where your application can access the resource that's associated with this service instance. b. For **IPv6 address**, provide an IPv6 IP address, if any, where your applications can access the resource that's associated with this service instance. c. For **Port**, specify any port your application must include to access the resource that's

Instance type	Steps
	associated with this service instance. **Port** is required when the service includes an SRV record or an Amazon Route 53 health check. d. (Optional) Under **Custom attributes**, specify any key-value pairs you want to associate with the resource.
EC2 instance	a. For **EC2 instance ID**, select the ID of the Amazon EC2 instance that you want to register as a AWS Cloud Map service instance. b. (Optional) Under **Custom attributes**, specify any key-value pairs you want to associate with the resource.

Instance type	Steps
Identifying information for another resource	a. Under **Standard attributes**, if the service configuration includes a **CNAME** DNS record, you'll see a **CNAME** field. For **CNAME**, specify the domain name that you want Route 53 to return in response to DNS queries (for example, `example.com`). b. Under **Custom attributes**, specify any identifying information for a resource that isn't an IP address or an Amazon EC2 instance ID as a key-value pair. For example, you can register a Lambda function by specifying a key called `function` and providing the name of the Lambda function as a value. You can also specify a key called `name` and provide a name that you can use for programmatic instance discovery.

9. Choose **Register service instance**.

AWS CLI

- When you submit a `RegisterInstance` request:

- For each DNS record that you define in the service that's specified by `ServiceId`, a record is created or updated in the hosted zone that's associated with the corresponding namespace.
- If the service includes `HealthCheckConfig`, a health check is created based on the settings in the health check configuration.
- Any health checks are associated with each of the new or updated records.

Register a service instance with the `register-instance` command (replace the *red* values with your own).

```
aws servicediscovery register-instance \
    --service-id srv-xxxxxxxxx \
    --instance-id myservice-xx \
    --attributes=AWS_INSTANCE_IPV4=172.2.1.3,AWS_INSTANCE_PORT=808
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. When you submit a `RegisterInstance` request:

   - For each DNS record that you define in the service that's specified by `ServiceId`, a record is created or updated in the hosted zone that's associated with the corresponding namespace.
   - If the service includes `HealthCheckConfig`, a health check is created based on the settings in the health check configuration.
   - Any health checks are associated with each of the new or updated records.

   Register a service instance with `register_instance()` (replace the *red* values with your own).

```
response = client.register_instance(
    Attributes={
        'AWS_INSTANCE_IPV4': '172.2.1.3',
        'AWS_INSTANCE_PORT': '808',
    },
    InstanceId='myservice-xx',
    ServiceId='srv-xxxxxxxxx',
)
# If you want to see the response
print(response)
```

Example response output

```
{
    'OperationId': '4yejorelbukcjzpnr6tlmrghsjwpngf4-k95yg2u7',
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Listing AWS Cloud Map service instances

To view a list of the service instances that you registered using a service, perform the following procedure.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. Choose the name of the namespace that contains the service for which you want to list service instances.
4. Choose the name of the service that you used to create the service instances. You'll see a list of instances under **Service instances**. You can enter the instance ID in the search field to list a specific instance. The **Created by** field shows the ID of the AWS account that registered the instance.

> **Note**
>
> If the namespace in which the instance is registered is a shared namespace, the AWS account ID under **Created by** may not be the same as your account ID. For more information about shared namespaces, see Shared AWS Cloud Map namespaces.

AWS CLI

- List service instances with the `list-instances` command (replace the *red* value with your own).

```
aws servicediscovery list-instances --service-id srv-xxxxxxxxx
```

AWS SDK for Python (Boto3)

1. If you don't already have Boto3 installed, you can find instructions for installing, configuring, and using Boto3 here.
2. Import Boto3 and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. List service instances with `list_instances()` (replace the *red* value with your own).

```
response = client.list_instances(
    ServiceId='srv-xxxxxxxxx',
)
# If you want to see the response
print(response)
```

Example response output

```
{
    'Instances': [
        {
            'Attributes': {
```

```
                'AWS_INSTANCE_IPV4': '172.2.1.3',
                'AWS_INSTANCE_PORT': '808',
            },
            'Id': 'i-xxxxxxxxxxxxxxxx',
        },
    ],
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Updating an AWS Cloud Map service instance

You can update service instances in two ways, depending on which values you want to update:

- **Update any values**: If you want to update any of the values that you specified for a service instance when you registered it, including custom attributes, you need to reregister the service instance and respecify all values. Follow the steps in Registering a resource as an AWS Cloud Map service instance, specifying the instance ID of the existing service instance for **Service instance ID**.

  Alternatively, you can use the `RegisterInstance` API. You can specify the ID of the existing instance and service using the `InstanceId` and `ServiceId` parameters and respecify other values.
- **Update only custom attributes**: If you want to update only the custom attributes for a service instance, you don't need to reregister the instance. You can update only those values. See Updating the custom attributes for a service instance.

Updating the custom attributes for a service instance

To update only custom attributes for a service instance

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. On the **Namespaces** page, choose the namespace that contains the service that you originally used to register the service instance.

4. On the **Namespace: *namespace-name*** page, choose the service that you used to register the service instance.
5. On the **Service: *service-name*** page, choose the name of the service instance that you want to update.
6. In the **Custom attributes** section, choose **Edit**.
7. On the **Edit service instance: *instance-name*** page, add, remove, or update custom attributes. You can update both keys and values for existing attributes.
8. Choose **Update service instance**.

Deregistering an AWS Cloud Map service instance

Before you can delete a service, you must deregister all service instances that were registered using the service.

To deregister a service instance, perform the following procedure.

AWS Management Console

1. Sign in to the AWS Management Console and open the AWS Cloud Map console at https://console.aws.amazon.com/cloudmap/.
2. In the navigation pane, choose **Namespaces**.
3. Choose the option for the namespace that contains the service instance that you want to deregister.
4. On the **Namespace: *namespace-name*** page, choose the service you used to register the service instance.
5. On the **Service: *service-name*** page, choose the service instance that you want to deregister.
6. Choose **Deregister**.
7. Confirm that you want to deregister the service instance.

AWS CLI

- Deregister a service instance with the `deregister-instance` command (replace the *red* values with your own). This command deletes the Amazon Route 53 DNS records and any health checks that AWS Cloud Map created for the specified instance.

```
aws servicediscovery deregister-instance \
    --service-id srv-xxxxxxxxx \
    --instance-id myservice-53
```

AWS SDK for Python (Boto3)

1. If you don't already have `Boto3` installed, you can find instructions for installing, configuring, and using `Boto3` here.
2. Import `Boto3` and use `servicediscovery` as your service.

```
import boto3
client = boto3.client('servicediscovery')
```

3. Deregister a service instance with `deregister-instance()` (replace the *red* values with your own). This command deletes the Amazon Route 53 DNS records and any health checks that AWS Cloud Map created for the specified instance.

```
response = client.deregister_instance(
    InstanceId='myservice-53',
    ServiceId='srv-xxxxxxxxx',
)
# If you want to see the response
print(response)
```

Example response output

```
{
    'OperationId': '4yejorelbukcjzpnr6tlmrghsjwpngf4-k98rnaiq',
    'ResponseMetadata': {
        '...': '...',
    },
}
```

Security in AWS Cloud Map

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that's built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS Cloud Map, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

The following documentation helps you understand how to apply the shared responsibility model when using AWS Cloud Map. The following topics show you how to configure AWS Cloud Map to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Cloud Map resources.

Topics

Identity and Access Management for AWS Cloud Map

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in)

and *authorized* (have permissions) to use AWS Cloud Map resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS Cloud Map identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS Cloud Map works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS Cloud Map)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Cloud Map works with IAM

Before you use IAM to manage access to AWS Cloud Map, learn what IAM features are available to use with AWS Cloud Map.

IAM feature	AWS Cloud Map support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No

IAM feature	AWS Cloud Map support
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how AWS Cloud Map and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS Cloud Map

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS Cloud Map

To view examples of AWS Cloud Map identity-based policies, see Identity-based policy examples for AWS Cloud Map.

Resource-based policies within AWS Cloud Map

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific

resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

> **Note**
>
> You can use AWS Resource Access Manager (AWS RAM) to securely share a AWS Cloud Map namespace. A resource-based policy is applied to your namespace by the AWS RAM service. For more information, see Shared AWS Cloud Map namespaces.

Policy actions for AWS Cloud Map

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS Cloud Map actions, see Actions defined by AWS Cloud Map in the *Service Authorization Reference*.

Policy actions in AWS Cloud Map use the following prefix before the action:

```
servicediscovery
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "servicediscovery:action1",
      "servicediscovery:action2"
```

```
        ]
```

To view examples of AWS Cloud Map identity-based policies, see Identity-based policy examples for AWS Cloud Map.

Policy resources for AWS Cloud Map

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS Cloud Map resource types and their ARNs, see Resources defined by AWS Cloud Map in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Cloud Map.

To view examples of AWS Cloud Map identity-based policies, see Identity-based policy examples for AWS Cloud Map.

Policy condition keys for AWS Cloud Map

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS Cloud Map condition keys, see Condition keys for AWS Cloud Map in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Cloud Map.

AWS Cloud Map supports the following service-specific condition keys that you can use to provide fine-grained filtering for your IAM policies.

`servicediscovery:NamespaceArn`

A filter that lets you get objects by specifying the Amazon Resource Name (ARN) for the related namespace.

`servicediscovery:NamespaceName`

A filter that lets you get objects by specifying the name of the related namespace.

`servicediscovery:ServiceArn`

A filter that lets you get objects by specifying the Amazon Resource Name (ARN) for the related service.

`servicediscovery:ServiceName`

A filter that lets you get objects by specifying the name of the related service.

`servicediscovery:ServiceCreatedByAccount`

A filter that lets you get objects by specifying the ID of the AWS account that created the service.

To view examples of AWS Cloud Map identity-based policies, see Identity-based policy examples for AWS Cloud Map.

ACLs in AWS Cloud Map

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AWS Cloud Map

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS Cloud Map

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Forward access sessions for AWS Cloud Map

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS Cloud Map

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS Cloud Map functionality. Edit service roles only when AWS Cloud Map provides guidance to do so.

Service-linked roles for AWS Cloud Map

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS Cloud Map

By default, users and roles don't have permission to create or modify AWS Cloud Map resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS Cloud Map, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Cloud Map in the *Service Authorization Reference*.

Topics

- AWS Cloud Map service instance example
- Create AWS Cloud Map service example
- Create AWS Cloud Map namespaces example

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Cloud Map resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS Cloud Map console

To access the AWS Cloud Map console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Cloud Map resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS Cloud Map console, also attach the AWS Cloud Map *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

AWS Cloud Map console access example

To grant full access to the AWS Cloud Map console, you grant the permissions in the following permissions policy:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "servicediscovery:*",
            "route53:GetHostedZone",
            "route53:ListHostedZonesByName",
            "route53:CreateHostedZone",
            "route53:DeleteHostedZone",
            "route53:ChangeResourceRecordSets",
            "route53:CreateHealthCheck",
            "route53:GetHealthCheck",
```

```
            "route53:DeleteHealthCheck",
            "route53:UpdateHealthCheck",
            "ec2:DescribeInstances",
            "ec2:DescribeVpcs",
            "ec2:DescribeRegions"
         ],
         "Resource":"*"
      }
   ]
}
```

Here's why the permissions are required:

`servicediscovery:*`

Lets you perform all AWS Cloud Map actions.

`route53:CreateHostedZone`, `route53:GetHostedZone`, `route53:ListHostedZonesByName`, `route53:DeleteHostedZone`

Lets AWS Cloud Map manage hosted zones when you create and delete public and private DNS namespaces.

`route53:CreateHealthCheck`, `route53:GetHealthCheck`, `route53:DeleteHealthCheck`, `route53:UpdateHealthCheck`

Lets AWS Cloud Map manage health checks when you include Amazon Route 53 health checks when you create a service.

`ec2:DescribeVpcs` and `ec2:DescribeRegions`

Let AWS Cloud Map manage private hosted zones.

Allow AWS Cloud Map users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow read access to all AWS Cloud Map resources

The following permissions policy grants the user read-only access to all AWS Cloud Map resources:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
```

```
            "servicediscovery:Get*",
            "servicediscovery:List*",
            "servicediscovery:DiscoverInstances"
         ],
         "Resource":"*"
      }
   ]
}
```

AWS Cloud Map service instance example

The following example shows a permissions policy that grants a user permission to register, deregister, and discover service instances. The `Sid`, or statement ID, is optional:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid" : "AllowInstancePermissions",
         "Effect": "Allow",
         "Action": [
            "servicediscovery:RegisterInstance",
            "servicediscovery:DeregisterInstance",
            "servicediscovery:DiscoverInstances",
            "servicediscovery:Get*",
            "servicediscovery:List*",
            "route53:GetHostedZone",
            "route53:ListHostedZonesByName",
            "route53:ChangeResourceRecordSets",
            "route53:CreateHealthCheck",
            "route53:GetHealthCheck",
            "route53:DeleteHealthCheck",
            "route53:UpdateHealthCheck",
            "ec2:DescribeInstances"
         ],
         "Resource": "*"
      }
   ]
}
```

The policy grants permissions to the actions that are required to register and manage service instances. The Route 53 permission is required if you're using public or private DNS namespaces because AWS Cloud Map creates, updates, and deletes Route 53 records and health checks when you register and deregister instances. The wildcard character (*) in `Resource` grants access to all AWS Cloud Map instances, and Route 53 records and health checks that are owned by the current AWS account.

Create AWS Cloud Map service example

When adding a permissions policy to allow an IAM identity to create a AWS Cloud Map service, you must specify the Amazon Resource Name (ARN) of both the AWS Cloud Map namespace and service in the resource field. The ARN includes the Region, account ID, and namespace ID. Since you won't know what the service ID of the service is yet, we recommend using a wildcard. The following is an example policy snippet.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "servicediscovery:CreateService"
            ],
            "Resource": [
                "arn:aws:servicediscovery:us-east-1:111122223333:namespace/ns-
p32123EXAMPLE",
                "arn:aws:servicediscovery:us-east-1:111122223333:service/*"
            ]
        }
    ]
}
```

Create AWS Cloud Map namespaces example

The following permissions policy allows users to create all types of AWS Cloud Map namespaces:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "servicediscovery:CreateHttpNamespace",
            "servicediscovery:CreatePrivateDnsNamespace",
            "servicediscovery:CreatePublicDnsNamespace",
            "route53:CreateHostedZone",
            "route53:GetHostedZone",
            "route53:ListHostedZonesByName",
            "ec2:DescribeVpcs",
            "ec2:DescribeRegions"
         ],
         "Resource":"*"
      }
   ]
}
```

AWS managed policies for AWS Cloud Map

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AWSCloudMapDiscoverInstanceAccess

You can attach `AWSCloudMapDiscoverInstanceAccess` to your IAM entities. Provides access to AWS Cloud Map Discovery API.

To view the permissions for this policy, see AWSCloudMapDiscoverInstanceAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSCloudMapReadOnlyAccess

You can attach `AWSCloudMapReadOnlyAccess` to your IAM entities. Grants read-only access to all AWS Cloud Map actions.

To view the permissions for this policy, see AWSCloudMapReadOnlyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSCloudMapRegisterInstanceAccess

You can attach `AWSCloudMapRegisterInstanceAccess` to your IAM entities. Grants read-only access to namespaces and services and grants permission to register and deregister service instances.

To view the permissions for this policy, see AWSCloudMapRegisterInstanceAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSCloudMapFullAccess

You can attach `AWSCloudMapFullAccess` to your IAM entities. Provides full access to all AWS Cloud Map actions

To view the permissions for this policy, see AWSCloudMapFullAccess in the *AWS Managed Policy Reference*.

AWS Cloud Map updates to AWS managed policies

View details about updates to AWS managed policies for AWS Cloud Map since this service began tracking these changes. For automatic alerts about changes, subscribe to the RSS feed on the AWS Cloud Map document history page.

Change	Description	Date
AWSCloudMapDiscoverInstanceAccess, AWSCloudMapRegisterInstanceAccess, AWSCloudMapReadOnlyAccess – Updates to existing policies.	AWS Cloud Map updated these policies to provide access to the new AWS Cloud Map `DiscoverInstanceRevision` API operations.	August 15, 2023

AWS Cloud Map API permissions reference

When you set up access control and write a permissions policy that you can attach to an IAM identity (identity-based policies), you can use the following list as a reference. The list includes each AWS Cloud Map API action and the actions that you must grant permissions access to. You specify the actions in the `Action` field for the policy. For details about the resource value you must specify in the `Resource` field or the IAM policy, see Actions, resources, and condition keys for AWS Cloud Map in the *Service Authorization Reference*.

You can use AWS Cloud Map–specific condition keys in your IAM policies for some operations. For more information, see Condition keys for AWS Cloud Map in the *Service Authorization Reference*.

To specify an action, use the `servicediscovery` prefix followed by the API action name, for example, `servicediscovery:CreatePublicDnsNamespace` and `route53:CreateHostedZone`.

Required permissions for AWS Cloud Map actions

CreateHttpNamespace

Required permissions (API action):

- `servicediscovery:CreateHttpNamespace`

CreatePrivateDnsNamespace

Required permissions (API action):

- `servicediscovery:CreatePrivateDnsNamespace`
- `route53:CreateHostedZone`

- route53:GetHostedZone
- route53:ListHostedZonesByName
- ec2:DescribeVpcs
- ec2:DescribeRegions

CreatePublicDnsNamespace

Required permissions (API action):

- servicediscovery:CreatePublicDnsNamespace
- route53:CreateHostedZone
- route53:GetHostedZone
- route53:ListHostedZonesByName

CreateService

Required Permissions (API Action): servicediscovery:CreateService

DeleteNamespace

Required permissions (API action):

- servicediscovery:DeleteNamespace

DeleteService

Required Permissions (API Action): servicediscovery:DeleteService

DeleteServiceAttributes

Required Permissions (API Action): servicediscovery:DeleteServiceAttributes

DeregisterInstance

Required permissions (API action):

- servicediscovery:DeregisterInstance
- route53:GetHealthCheck
- route53:DeleteHealthCheck
- route53:UpdateHealthCheck

DiscoverInstances

Required Permissions (API Action): servicediscovery:DiscoverInstances

GetInstance

Required Permissions (API Action): `servicediscovery:GetInstance`

GetInstancesHealthStatus

Required Permissions (API Action): `servicediscovery:GetInstancesHealthStatus`

GetNamespace

Required Permissions (API Action): `servicediscovery:GetNamespace`

GetOperation

Required Permissions (API Action): `servicediscovery:GetOperation`

GetService

Required Permissions (API Action): `servicediscovery:GetService`

GetServiceAttributes

Required Permissions (API Action): `servicediscovery:GetServiceAttributes`

ListInstances

Required Permissions (API Action): `servicediscovery:ListInstances`

ListNamespaces

Required Permissions (API Action): `servicediscovery:ListNamespaces`

ListOperations

Required Permissions (API Action): `servicediscovery:ListOperations`

ListServices

Required Permissions (API Action): `servicediscovery:ListServices`

ListTagsForResource

Required Permissions (API Action): `servicediscovery:ListTagsForResource`

RegisterInstance

Required permissions (API action):

- `servicediscovery:RegisterInstance`

- `route53:GetHealthCheck`
- `route53:CreateHealthCheck`
- `route53:UpdateHealthCheck`
- `ec2:DescribeInstances`

TagResource

Required Permissions (API Action): `servicediscovery:TagResource`

UntagResource

Required Permissions (API Action): `servicediscovery:UntagResource`

UpdateHttpNamespace

Required Permissions (API Action): `servicediscovery:UpdateHttpNamespace`

UpdateInstanceCustomHealthStatus

Required Permissions (API Action): `servicediscovery:UpdateInstanceCustomHealthStatus`

UpdatePrivateDnsNamespace

Required permissions (API action):

- `servicediscovery:UpdatePrivateDnsNamespace`
- `route53:ChangeResourceRecordSets`

UpdatePublicDnsNamespace

Required permissions (API action):

- `servicediscovery:UpdatePublicDnsNamespace`
- `route53:ChangeResourceRecordSets`

UpdateService

Required permissions (API action):

- `servicediscovery:UpdateService`
- `route53:GetHealthCheck`
- `route53:CreateHealthCheck`
- `route53:DeleteHealthCheck`

- `route53:UpdateHealthCheck`

[UpdateServiceAttributes]

Required Permissions (API Action): `servicediscovery:UpdateServiceAttributes`

Troubleshooting AWS Cloud Map identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Cloud Map and IAM.

Topics

I am not authorized to perform an action in AWS Cloud Map

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `servicediscovery:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 servicediscovery:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `servicediscovery:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS Cloud Map.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS Cloud Map. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS Cloud Map resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS Cloud Map supports these features, see How AWS Cloud Map works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS Cloud Map

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS Cloud Map

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

AWS Cloud Map is primarily a global service. However, you can use AWS Cloud Map to create Route 53 health checks that check the health of resources in specific Regions, such as Amazon EC2 instances and Elastic Load Balancing load balancers.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in AWS Cloud Map

As a managed service, AWS Cloud Map is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS Cloud Map through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

You can improve the security posture of your VPC by configuring AWS Cloud Map to use an interface VPC endpoint. For more information, see Access AWS Cloud Map using an interface endpoint (AWS PrivateLink).

Access AWS Cloud Map using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS Cloud Map. You can access AWS Cloud Map as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS Cloud Map.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for AWS Cloud Map.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for AWS Cloud Map

Before you set up an interface endpoint for AWS Cloud Map, review Considerations in the *AWS PrivateLink Guide*.

If your Amazon VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the Amazon CloudWatch Logs User Guide.

VPC endpoints don't support AWS cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to AWS Cloud Map.

VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the Amazon VPC User Guide.

The security group attached to the VPC endpoint must allow incoming connections on port 443 from the private subnet of the Amazon VPC.

Create an interface endpoint for AWS Cloud Map

You can create an interface endpoint for AWS Cloud Map using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for AWS Cloud Map using the following service names:

> **Note**
>
> `DiscoverInstances` API won't be available over these two endpoints.

```
com.amazonaws.region.servicediscovery
```

```
com.amazonaws.region.servicediscovery-fips
```

Create an interface endpoint for AWS Cloud Map data plane to access the `DiscoverInstances` API using the following service names:

```
com.amazonaws.region.data-servicediscovery
```

```
com.amazonaws.region.data-servicediscovery-fips
```

> **Note**
>
> You'll need to disable host prefix injection when you call `DiscoverInstances` with the regional or zonal VPCE DNS names for data plane endpoints. The AWS CLI and AWS SDKs prepend the service endpoint with various host prefixes when you call each API operation, which produces invalid URLS when you specify a VPC endpoint.

If you enable private DNS for the interface endpoint, you can make API requests to AWS Cloud Map using its default Regional DNS name. For example, `servicediscovery.us-east-1.amazonaws.com`.

VPCE AWS PrivateLink connection is supported in any Region where AWS Cloud Map is supported; however, a customer needs to check which Availability Zones support VPCE before defining an endpoint. To find out which Availability Zones are supported with interface VPC endpoints in a Region, use the describe-vpc-endpoint-services command or use the AWS Management Console. For example, the following commands return the availability zones to which you can deploy an AWS Cloud Map interface VPC endpoints within the US East (Ohio) Region:

```
aws --region us-east-2 ec2 describe-vpc-endpoint-services --query 'ServiceDetails[?
ServiceName==`com.amazonaws.us-east-2.servicediscovery`].AvailabilityZones[]'
```

Monitoring AWS Cloud Map

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. However, before you start monitoring, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

Topics

Log AWS Cloud Map API calls using AWS CloudTrail

AWS Cloud Map is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for AWS Cloud Map as events. The calls captured include calls from the AWS Cloud Map console and code calls to the AWS Cloud Map API operations. Using the information collected by CloudTrail, you can determine the request that was made to AWS Cloud Map, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

AWS Cloud Map data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, discovering a registered instance in a namespace). These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn't log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the AWS Cloud Map resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. For more information about how to log data events, see Logging data events with the AWS Management Console and Logging data events with the AWS Command Line Interface in the *AWS CloudTrail User Guide*.

The following table lists the AWS Cloud Map resource types for which you can log data events. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
AwsApiCall	`AWS::ServiceDiscovery::Namespace`	• DiscoverInstances • DiscoverInstancesRevision
AwsApiCall	`AWS::ServiceDiscovery::Service`	• DiscoverInstances • DiscoverInstancesRevision • GetServiceAttributes

You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector in the *AWS CloudTrail API Reference*.

The following example shows how to configure advanced event selectors to log all AWS Cloud Map data events.

```
"AdvancedEventSelectors":
[
    {
        "Name": "Log all AWS Cloud Map data events",
        "FieldSelectors": [
            { "Field": "eventCategory", "Equals": ["Data"] },
            { "Field": "resources.type", "Equals":
 ["AWS::ServiceDiscovery::Namespace"] }
        ]
    }
]
```

AWS Cloud Map management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

AWS Cloud Map logs all AWS Cloud Map control plane operations as management events. For a list of the AWS Cloud Map control plane operations that AWS Cloud Map logs to CloudTrail, see the AWS Cloud Map API Reference.

AWS Cloud Map event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail management event that demonstrates the `CreateHTTPNamespace` operation.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:alejandro_rosalez",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alejandro_rosalez",
        "accountId": "111122223333",
        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "sessionContext": {
```

```
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/readonly-role",
                "accountId": "111122223333",
                "userName": "alejandro_rosalez"
            },
            "attributes": {
                "creationDate": "2024-03-19T16:15:37Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-03-19T19:23:13Z",
    "eventSource": "servicediscovery.amazonaws.com",
    "eventName": "CreateHttpNamespace",
    "awsRegion": "eu-west-3",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36",
    "requestParameters": {
        "name": "example-namespace",
        "creatorRequestId": "eda8b524-ca14-4f68-a176-dc4dfd165c26",
        "tags": []
    },
    "responseElements": {
        "operationId": "7xm4i7ghhkaalma666nrg6itf2eylcbp-gwipo38o"
    },
    "requestID": "641274d0-dbbe-4e64-9b53-685769a086c7",
    "eventID": "4a1ab076-ef1b-4bcf-aa95-cec5fb64f2bd",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "servicediscovery.eu-west-3.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

The following example shows a CloudTrail data event that demonstrates the `DiscoverInstances` operation.


```
{
            "eventVersion": "1.09",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE:alejandro_rosalez",
                "arn": "arn:aws:sts::111122223333:assumed-role/role/Admin",
                "accountId": "111122223333",
                "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AROA123456789EXAMPLE",
                        "arn": "arn:aws:iam::"111122223333":role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "attributes": {
                        "creationDate": "2024-03-19T16:15:37Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2024-03-19T21:19:12Z",
            "eventSource": "servicediscovery.amazonaws.com",
            "eventName": "DiscoverInstances",
            "awsRegion": "eu-west-3",
            "sourceIPAddress": "13.38.34.79",
            "userAgent": "Boto3/1.20.34 md/Botocore#1.34.60 ua/2.0 os/linux#6.5.0-1014-
aws md/arch#x86_64 lang/python#3.10.12 md/pyimpl#CPython cfg/retry-mode#legacy
 Botocore/1.34.60",
            "requestParameters": {
                "namespaceName": "example-namespace",
                "serviceName": "example-service",
                "queryParameters": {"example-key": "example-value"}
            },
            "responseElements": null,
            "requestID": "e5ee36f1-edb0-4814-a4ba-2e8c97621c79",
            "eventID": "503cedb6-9906-4ee5-83e0-a64dde27bab0",
            "readOnly": true,
            "resources": [
```

```
                {
                    "accountId": "111122223333",
                    "type": "AWS::ServiceDiscovery::Namespace",
                    "ARN": "arn:aws:servicediscovery:eu-west-3:111122223333:namespace/
ns-vh4nbmhEXAMPLE"
                },
                {
                    "accountId": "111122223333",
                    "type": "AWS::ServiceDiscovery::Service",
                    "ARN": "arn:aws:servicediscovery:eu-west-3:111122223333:service/
srv-h46op6ylEXAMPLE"
                }
            ],
            "eventType": "AwsApiCall",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data",
            "tlsDetails": {
                "tlsVersion": "TLSv1.3",
                "cipherSuite": "TLS_AES_128_GCM_SHA256",
                "clientProvidedHostHeader": "data-servicediscovery.eu-
west-3.amazonaws.com"
            },
            "sessionCredentialFromConsole": "true"
        }
```

For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Tagging your AWS Cloud Map resources

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your AWS Cloud Map services to help you track each service's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to AWS Cloud Map and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value.

You can work with tags using the AWS Management Console, the AWS CLI, and the AWS Cloud Map API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

How resources are tagged

You can tag new or existing AWS Cloud Map namespaces and services.

If you're using the AWS Cloud Map console, you can apply tags to new resources when they are created or to existing resources at any time using the **Tags** tab on the relevant resource page.

If you're using the AWS Cloud Map API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action or to existing resources using the TagResource API action. For more information, see TagResource.

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, the resource creation process fails. This

ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after resource creation.

The following table describes the AWS Cloud Map resources that can be tagged, and the resources that can be tagged on creation.

Tagging support for AWS Cloud Map resources

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (AWS Cloud Map API, AWS CLI, AWS SDK)
AWS Cloud Map namespaces	Yes	No. Namespace tags don't propagate to any other resources associated with the namespace.	Yes
AWS Cloud Map services	Yes	No. Service tags don't propagate to any other resources associated with the service.	Yes

Restrictions

The following basic restrictions apply to tags:

- Maximum number of tags for each resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services might have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . _ : / @.

- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values, as it is reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags-per-resource limit.

Updating tags for AWS Cloud Map resources

Use the following AWS CLI commands or AWS Cloud Map API operations to add, update, list, and delete the tags for your resources.

Tagging support for AWS Cloud Map resources

Task	API action	AWS CLI	AWS Tools for Windows PowerShell
Add or overwrite one or more tags.	TagResource	tag-resource	Add-SDResourceTag
Delete one or more tags.	UntagResource	untag-resource	Remove-SDResourceTag
List tags for a resource	ListTagsForResource	list-tags-for-resource	Get-SDResourceTag

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing resource

The following command tags an existing resource.

```
aws servicediscovery tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing resource

The following command deletes a tag from an existing resource.

```
aws servicediscovery untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws servicediscovery list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

Task	API action	AWS CLI	AWS Tools for Windows PowerShell
Create an HTTP namespace	CreateHttpNamespace	create-http-namespace	New-SDHttpNamespace
Create a private namespace based on DNS	CreatePrivateDnsNamespace	create-private-dns-namespace	New-SDPrivateDnsNamespace
Create a public namespace based on DNS	CreatePublicDnsNamespace	create-public-dns-namespace	New-SDPublicDnsNamespace
Create a service	CreateService	create-service	New-SDService

AWS Cloud Map service quotas

AWS Cloud Map resources are subject to the following account-level service quotas. Each quota listed applies to each AWS Region where you create AWS Cloud Map resources.

Name	Default	Adjustable	Description
Custom attributes per instance	Each supported Region: 30	No	The maximum number of custom attributes that you can specify when you register an instance.
DiscoverInstances operation per account burst rate	Each supported Region: 2,000	Yes	The maximum burst rate to call DiscoverInstances operation from a single account.
DiscoverInstances operation per account steady rate	Each supported Region: 1,000	Yes	The maximum steady rate to call DiscoverInstances operation from a single account.
DiscoverInstancesRevision operation per account rate	Each supported Region: 3,000	Yes	The maximum rate to call DiscoverInstancesRevision operation from a single account.
Instances per namespace	Each supported Region: 2,000	Yes	The maximum number of service instances that you can register using the same namespace.
Instances per service	Each supported Region: 1,000	No	The maximum number of instances that you can register in a Region using the same service.

Name	Default	Adjustable	Description
Namespaces per Region	Each supported Region: 50	Yes	The maximum number of namespaces that you can create per Region.

* When you create a namespace, we automatically create an Amazon Route 53 hosted zone. This hosted zone counts against the quota on the number of hosted zones that you can create with an AWS account. For more information, see Quotas on hosted zones in the *Amazon Route 53 Developer Guide*.

** Increasing the instances for DNS namespaces for AWS Cloud Map requires an increase to the records per hosted zone Route 53 limit, which incurs additional charges.

Managing your AWS Cloud Map service quotas

AWS Cloud Map has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see What is Service Quotas? in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your AWS Cloud Map service quotas.

AWS Management Console

To view AWS Cloud Map service quotas using the AWS Management Console

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **AWS Cloud Map**.
4. In the service quotas list for AWS Cloud Map, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

   To view additional information about a service quota, such as the description, choose the quota name to bring up the quota details.
5. (Optional) To request a quota increase, select the quota that you want to increase and choose **Request increase at account-level**.

To work more with service quotas using the AWS Management Console see the Service Quotas User Guide.

AWS CLI

To view AWS Cloud Map service quotas using the AWS CLI

Run the following command to view the default AWS Cloud Map quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code AWSCloudMap \
    --output table
```

Run the following command to view your applied AWS Cloud Map quotas.

```
aws service-quotas list-service-quotas \
    --service-code AWSCloudMap
```

For more information about working with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference. To request a quota increase, see the `request-service-quota-increase` command in the AWS CLI Command Reference.

Handle AWS Cloud Map DiscoverInstances API request throttling

AWS Cloud Map throttles DiscoverInstances API requests for each AWS account on a per-Region basis. Throttling helps improve the performance of the service and helps provide fair usage for all AWS Cloud Map customers. Throttling ensures that calls to the AWS Cloud Map DiscoverInstances API doesn't exceed the maximum allowed DiscoverInstances API request quotas. DiscoverInstances API calls originating from any of the following sources are subject to the request quotas:

- A third-party application
- A command line tool
- The AWS Cloud Map console

If you exceed an API throttling quota, you get the `RequestLimitExceeded` error code. For more information, see the section called “Request rate limiting”.

How throttling is applied

AWS Cloud Map uses the token bucket algorithm to implement API throttling. With this algorithm, your account has a *bucket* that holds a specific number of *tokens*. The number of tokens in the bucket represents your throttling quota at any given second. There is one bucket for a single Region, and it applies to all endpoints in the Region.

Request rate limiting

Throttling limits the number of DiscoverInstances API requests that you can make. Each request removes one token from the bucket. For example, the bucket size for the DiscoverInstances API operation is 2,000 tokens, so you can make up to 2,000 DiscoverInstances requests in one second. If you exceed 2,000 requests in one second, you're throttled and the remaining requests within that second fail.

Buckets automatically refill at a set rate. If the bucket isn't at capacity, a set number of tokens is added back every second until the bucket reaches capacity. If the bucket is at capacity when refill tokens arrive, then these tokens are discarded. The bucket size for the DiscoverInstances API operation is 2,000 tokens, and the refill rate is 1,000 tokens every second. If you make 2,000 DiscoverInstances API requests in a second, the bucket is immediately reduced to zero (0) tokens. The bucket is then refilled by up to 1,000 tokens every second until it reaches its maximum capacity of 2,000 tokens.

You can use tokens as they are added to the bucket. You don't need to wait for the bucket to be at maximum capacity before you make API requests. If you deplete the bucket by making 2,000 DiscoverInstances API requests in one second, you can still make up to 1,000 DiscoverInstances API requests every second after that for as long as you need. This means that you can immediately use the refill tokens as they are added to your bucket. The bucket only starts to refill to the maximum capacity when you make fewer API requests every second than the refill rate.

Retries or batch processing

If an API request fails, your application might need to retry the request. To reduce the number of API requests, use an appropriate sleep interval between successive requests. For best results, use an increasing or variable sleep interval.

Calculating the sleep interval

When you have to poll or retry an API request, we recommend using an exponential backoff algorithm to calculate the sleep interval between API calls. By using progressively longer wait times between retries for consecutive error responses, you can reduce the number of failed requests. For more information and implementation examples of this algorithm, see Retry Behavior in the *AWS SDKs and Tools Reference Guide*.

Adjusting API throttling quotas

You can request an increase to API throttling quotas for your AWS account. To request a quota adjustment, contact the AWS Support Center.

Document history for AWS Cloud Map

The following table describes the major updates and new features for the *AWS Cloud Map Developer Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
AWS Cloud Map cross-account namespace sharing	You can now share namespaces with other AWS accounts or within an organization in AWS Organizations using AWS Resource Access Manager (AWS RAM) for simplifie d cross-account service discovery and registry.	August 14, 2025
AWS Cloud Map service attributes	You can now specify attribute s at the service level to avoid duplicating attributes across instances that are registered to a service. You can use these attributes for complex traffic routing, setting timeout and retry values, and for coordinat ion between services and external integrations.	December 13, 2024
Tutorials added	Two tutorials showing common use cases for using AWS Cloud Map added.	March 27, 2024
CloudTrail integration documentation updated	The documentation describin g the AWS Cloud Map integration with CloudTrail	March 20, 2024

	to log API activity has been updated.	
Managed policy updates	`AWSCloudMapDiscoverInstanceAccess`, `AWSCloudMapRegisterInstanceAccess`, and `AWSCloudMapReadOnlyAccess` policies were updated.	September 20, 2023
Cloud Map and AWS PrivateLink	You can now use an AWS PrivateLink to create a private connection between your VPC and AWS Cloud Map.	September 15, 2023
Managed policy update	`AWSCloudMapDiscoverInstanceAccess` policy was updated.	August 15, 2023
AWS SDK for Python	Added Python command line examples.	September 13, 2022
IPv6 support	API endpoints are now available in `IPv6`-only networks.	January 28, 2022
Service instance discovery	AWS Cloud Map added support for creating services in a namespace that supports DNS queries that are discoverable only using the DiscoverInstances API operation and not using DNS queries.	March 24, 2021

Resource tagging	AWS Cloud Map added support for adding metadata tags to your namespaces and services using the AWS Management Console.	February 8, 2021
Resource tagging	AWS Cloud Map added support for adding metadata tags to your namespaces and services using the AWS CLI and APIs.	June 22, 2020
Initial Release	This is the first release of *AWS Cloud Map Developer Guide*.	November 28, 2018