# Security in Amazon Bedrock

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 Amazon Bedrock, 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.

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

**Topics**

# Data protection

The AWS shared responsibility model applies to data protection in Amazon Bedrock. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon Bedrock or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

**Data protection in Amazon Bedrock**

Amazon Bedrock doesn't use your prompts and continuations to train any AWS models or distribute them to third parties.

Each model provider has an escrow account that they upload their models to. The Amazon Bedrock inference account has permissions to call these models, but the escrow accounts themselves don't have outbound permissions to Amazon Bedrock accounts. Additionally, model providers don't have access to Amazon Bedrock logs or access to customer prompts and continuations.

Amazon Bedrock doesn’t store or log your data in its service logs.

**Data protection in Amazon Bedrock model customization**

Your training data isn't used to train the base Titan models or distributed to third parties. Other usage data, such as usage timestamps, logged account IDs, and other information logged by the service, is also not used to train the models.

Amazon Bedrock uses the fine tuning data you provide only for fine tuning an Amazon Bedrock foundation model. Amazon Bedrock doesn't use fine tuning data for any other purpose, such as training base foundation models.

Bedrock uses your training data with the CreateModelCustomizationJob action, or with the console, to create a custom model which is a fine tuned version of an Amazon Bedrock foundational model. Your custom models are managed and stored by AWS. By default, custom models are encrypted with AWS Key Management Service keys that AWS owns, but you can use your own AWS KMS keys to encrypt your custom models. You encrypt a custom model when you submit a fine tuning job with the console or programmatically with the `CreateModelCustomizationJob` action.

None of the training or validation data you provide for fine tuning is stored in Amazon Bedrock accounts, once the fine tuning job completes. During training, your data exists in AWS Service Management Connector instance memory, but is encrypted on these machines using an XTS-AES-256 cipher that is implemented on a hardware module, on the instance itself.

We don't recommend using confidential data to train a custom model as the model might generate inference responses based on that confidential data. If you use confidential data to train a custom model, the only way to prevent responses based on that data is to delete the custom model, remove the confidential data from your training dataset, and retrain the custom model.

Custom model metadata (name and Amazon Resource Name) and a provisioned model's metadata is stored in an Amazon DynamoDB table that is encrypted with a key that the Amazon Bedrock service owns.

**Topics**

- Protect your data using Amazon VPC and AWS PrivateLink

## Data encryption

Amazon Bedrock uses encryption to protect data at rest and data in transit.

**Topics**

### Encryption in transit

Within AWS, all inter-network data in transit supports TLS 1.2 encryption.

Requests to the Amazon Bedrock API and console are made over a secure (SSL) connection. You pass AWS Identity and Access Management (IAM) roles to Amazon Bedrock to provide permissions to access resources on your behalf for training and deployment.

### Encryption at rest

Amazon Bedrock provides Encryption of model customization jobs and artifacts at rest.

### Key management

Use the AWS Key Management Service to manage the keys that you use to encrypt your resources. For more information, see AWS Key Management Service concepts. You can encrypt the following resources with a KMS key.

- Through Amazon Bedrock
  - Model customization jobs and their output custom models – During job creation in the console or by specifying the `customModelKmsKeyId` field in the CreateModelCustomizationJob API call.

  - Agents – During agent creation in the console or by specifying the  field in the CreateAgent API call.
  - Data source ingestion jobs for knowledge bases – During knowledge base creation in the console or by specifying the `kmsKeyArn` field in the CreateDataSource or UpdateDataSource API call.
  - Vector stores in Amazon OpenSearch Service – During vector store creation. For more information, see Creating, listing, and deleting Amazon OpenSearch Service collections and Encryption of data at rest for Amazon OpenSearch Service.
- Through Amazon S3 – For more information, see Using server-side encryption with AWS KMS keys (SSE-KMS).
  - Training, validation, and output data for model customization
  - Data sources for knowledge bases
- Through AWS Secrets Manager – For more information, see Secret encryption and decryption in AWS Secrets Manager
  - Vector stores for third-party models

After you encrypt a resource, you can find the ARN of the KMS key by selecting a resource and viewing its **Details** in the console or by using the following `Get` API calls.

- GetModelCustomizationJob
- GetAgent
- GetIngestionJob

## Encryption of model customization jobs and artifacts

By default, Amazon Bedrock encrypts the following model artifacts from your model customization jobs with an AWS managed key.

- The model customization job
- The output files (training and validation metrics) from the model customization job
- The resulting custom model

Optionally, you can encrypt the model artifacts by creating a customer managed key. For more information about AWS KMS keys, see Customer managed keys in the *AWS Key Management Service Developer Guide*. To use a customer managed key, carry out the following steps.

1. Create a customer managed key with the AWS Key Management Service.
2. Attach a resource-based policy with permissions for the specified-roles to create or use custom models.

**Topics**

### Create a customer managed key

First ensure that you have `CreateKey` permissions. Then follow the steps at Creating keys to create a customer managed key either in the AWS KMS console or the CreateKey API operation. Make sure to create a symmetric encryption key.

Creation of the key returns an `Arn` for the key that you can use as the `customModelKmsKeyId` when submitting a model customization job.

### Create a key policy and attach it to the customer managed key

Attach the following resource-based policy to the KMS key by following the steps at Creating a key policy. The policy contains two statements.

1. Permissions for a role to encrypt model customization artifacts. Add ARNs of custom model builder roles to the `Principal` field.
2. Permissions for a role to use a custom model in inference. Add ARNs of custom model user roles to the `Principal` field.

```
{
    "Version": "2012-10-17",
    "Id": "KMS Key Policy",
    "Statement": [
        {
            "Sid": "Permissions for custom model builders",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/role"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:DescribeKey",
                "kms:CreateGrant"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Permissions for custom model users",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/role"
            },
            "Action": "kms:Decrypt",
            "Resource": "*"
        }
}
```

**Encryption of training, validation, and output data**

When you use Amazon Bedrock to run a model customization job, you store the input (training/validation data) files in your Amazon S3 bucket. When the job completes, Amazon Bedrock stores the output metrics files in the S3 bucket that you specifed when creating the job and the resulting custom model artifacts in an Amazon S3 bucket controlled by AWS.

The input and output files are encrypted with Amazon S3 SSE-S3 server-side encryption by default, using an *AWS managed key*. This type of key is created, managed, and used on your behalf by AWS.

You can instead choose to encrypt these files with a *customer managed key* that you create, own, and manage yourself. Refer to the preceding sections and the following links to learn how to create customer managed keys and key policies.

- To learn more about Amazon S3 SSE-S3 server-side encryption, see Using server-side encryption with Amazon S3 managed keys (SSE-S3)
- To learn more about customer managed keys for encrypting S3 objects, see Using server-side encryption with AWS KMS keys (SSE-KMS)

## Encryption of agent resources

Amazon Bedrock encrypts your agent's session information. By default, Amazon Bedrock encrypts this data using an AWS managed key. Optionally, you can encrypt the agent artifacts using a customer managed key.

For more information about AWS KMS keys, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

If you encrypt sessions with your agent with a custom KMS key, you must set up the following identity-based policy and resource-based policy to allow Amazon Bedrock to encrypt and decrypt agent resources on your behalf.

1. Attach the following identity-based policy to an IAM role or user with permissions to make `InvokeAgent` calls. This policy validates the user making an `InvokeAgent` call has KMS permissions. Replace the *${region}*, *${account-id}*, *${agent-id}*, and *${key-id}* with the appropriate values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Amazon Bedrock to encrypt and decrypt Agent resources on
 behalf of authorized users",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:${region}:${account-id}:key/${key-id}",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:bedrock:arn":
 "arn:aws:bedrock:${region}:${account-id}:agent/${agent-id}"
                }
            }
        }
    ]
}
```

2. Attach the following resource-based policy to your KMS key. Change the scope of the permissions as necessary. Replace the *${region}*, *${account-id}*, *${agent-id}*, and *${key-id}* with the appropriate values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow account root to modify the KMS key, not used by Amazon
 Bedrock.",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::${account-id}:root"
            },
            "Action": "kms:*",
            "Resource": "arn:aws:kms:${region}:${account-id}:key/${key-id}"
        },
        {
            "Sid": "Allow Amazon Bedrock to encrypt and decrypt Agent resources on
 behalf of authorized users",
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:${region}:${account-id}:key/${key-id}",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:bedrock:arn":
 "arn:aws:bedrock:${region}:${account-id}:agent/${agent-id}"
                }
            }
        },
        {
            "Sid": "Allow the service role to use the key to encrypt and decrypt
 Agent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::${account-id}:role/
AmazonBedrockExecutionRoleForAgents_${suffix}"
```

```
            },
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt",
            ],
            "Resource": "arn:aws:kms:${region}:${account-id}:key/${key-id}"
},
        {
           "Sid": "Allow the attachment of persistent resources",
           "Effect": "Allow",
           "Principal": {
               "Service": "bedrock.amazonaws.com"
           },
           "Action": [
               "kms:CreateGrant",
               "kms:ListGrants",
               "kms:RevokeGrant"
           ],
           "Resource": "*",
           "Condition": {
               "Bool": {
                   "kms:GrantIsForAWSResource": "true"
               }
           }
         }
    ]
}
```

## Encryption of knowledge base resources

Amazon Bedrock encrypts resources related to your knowledge bases. By default, Amazon Bedrock encrypts this data using an AWS managed key. Optionally, you can encrypt the model artifacts using a customer managed key.

Encryption with a KMS key can occur with the following processes:

- Transient data storage while ingesting your data sources
- Passing information to OpenSearch Service if you let Amazon Bedrock set up your vector database
- Querying a knowledge base

The following resources used by your knowledge bases can be encrypted with a KMS key. If you encrypt them, you need to add permissions to decrypt the KMS key.

- Data sources stored in an Amazon S3 bucket
- Third-party vector stores

For more information about AWS KMS keys, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

**Topics**

### Encryption of transient data storage during data ingestion

When you set up a data ingestion job for your knowledge base, you can encrypt the job with a custom KMS key.

To allow the creation of a AWS KMS key for transient data storage in the process of ingesting your data source, attach the following policy to your Amazon Bedrock service role. Replace the *region*, *account-id*, and *key-id* with the appropriate values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "kms:GenerateDataKey",
              "kms:Decrypt"
          ],
          "Resource": [
              "arn:aws:kms:region:account-id:key/key-id"
          ]
        }
    ]
}
```

### Encryption of information passed to Amazon OpenSearch Service

If you opt to let Amazon Bedrock create a vector store in Amazon OpenSearch Service for your knowledge base, Amazon Bedrock can pass a KMS key that you choose to Amazon OpenSearch Service for encryption. To learn more about encryption in Amazon OpenSearch Service, see Encryption in Amazon OpenSearch Service.

### Encryption of knowledge base retrieval

You can encrypt sessions in which you generate responses from querying a knowledge base with a KMS key. To do so, include the ARN of a KMS key in the `kmsKeyArn` field when making a RetrieveAndGenerate request. Attach the following policy, replacing the *`values`* appropriately to allow Amazon Bedrock to encrypt the session context.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:region:account-id:key/key-id
        }
    ]
}
```

### Permissions to decrypt your AWS KMS key for your data sources in Amazon S3

You store the data sources for your knowledge base in your Amazon S3 bucket. To encrypt these documents at rest, you can use the Amazon S3 SSE-S3 server-side encryption option. With this option, objects are encrypted with service keys managed by the Amazon S3 service.

For more information, see Protecting data using server-side encryption with Amazon S3-managed encryption keys (SSE-S3) in the *Amazon Simple Storage Service User Guide*.

If you encrypted your data sources in Amazon S3 with a custom AWS KMS key, attach the following policy to your Amazon Bedrock service role to allow Amazon Bedrock to decrypt your key. Replace *region* and *account-id* with the region and account ID to which the key belongs. Replace *key-id* with the ID of your AWS KMS key.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "KMS:Decrypt",
        ],
        "Resource": [
            "arn:aws:kms:region:account-id:key/key-id"
        ],
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.region.amazonaws.com"
                ]
            }
        }
    }]
}
```

**Permissions to decrypt an AWS Secrets Manager secret for the vector store containing your knowledge base**

If the vector store containing your knowledge base is configured with an AWS Secrets Manager secret, you can encrypt the secret with a custom AWS KMS key by following the steps at Secret encryption and decryption in AWS Secrets Manager.

If you do so, you attach the following policy to your Amazon Bedrock service role to allow it to decrypt your key. Replace *region* and *account-id* with the region and account ID to which the key belongs. Replace *key-id* with the ID of your AWS KMS key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
            ]
        }
    ]
}
```

## Protect your data using Amazon VPC and AWS PrivateLink

To control access to your data, we recommend that you use a virtual private cloud (VPC) with Amazon VPC. Using a VPC protects your data and lets you monitor all network traffic in and out of the AWS job containers by using VPC Flow Logs. You can further protect your data by configuring your VPC so that your data isn't available over the internet and instead creating a VPC interface endpoint with AWS PrivateLink to establish a private connection to your data.

For an example of using VPC to protect data that you integrate with Amazon Bedrock see Protect model customization jobs using a VPC.

### Use interface VPC endpoints (AWS PrivateLink)

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

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 Amazon Bedrock.

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

#### Considerations for Amazon Bedrock VPC endpoints

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

Amazon Bedrock supports making the following API calls through VPC endpoints.

| Category | Endpoint prefix |
|---|---|
| Amazon Bedrock Control Plane API actions | `bedrock` |
| Amazon Bedrock Runtime API actions | `bedrock-runtime` |
| Agents for Amazon Bedrock Build-time API actions | `bedrock-agent` |
| Agents for Amazon Bedrock Runtime API actions | `bedrock-agent-runtime` |

**Availability Zones**

Amazon Bedrock and Agents for Amazon Bedrock endpoints are available in multiple Availability Zones.

**Create an interface endpoint for Amazon Bedrock**

You can create an interface endpoint for Amazon Bedrock 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 Amazon Bedrock using any of the following service names:

- `com.amazonaws.`*`region`*`.bedrock`
- `com.amazonaws.`*`region`*`.bedrock-runtime`
- `com.amazonaws.`*`region`*`.bedrock-agent`
- `com.amazonaws.`*`region`*`.bedrock-agent-runtime`

After you create the endpoint, you have the option to enable a private DNS hostname. Enable this setting by selecting Enable Private DNS Name in the VPC console when you create the VPC endpoint.

If you enable private DNS for the interface endpoint, you can make API requests to Amazon Bedrock using its default Regional DNS name. The following examples show the format of the default Regional DNS names.

- `bedrock.`*`region`*`.amazonaws.com`
- `bedrock-runtime.`*`region`*`.amazonaws.com`
- `bedrock-agent.`*`region`*`.amazonaws.com`
- `bedrock-agent-runtime.`*`region`*`.amazonaws.com`

### Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Amazon Bedrock through the interface endpoint. To control the access allowed to Amazon Bedrock from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

#### Example: VPC endpoint policy for Amazon Bedrock actions

The following is an example of a custom endpoint policy. When you attach this resource-based policy to your interface endpoint, it grants access to the listed Amazon Bedrock actions for all principals on all resources.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ],
         "Resource":"*"
      }
```

```
    ]
}
```

## Identity and access management for Amazon Bedrock

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 Amazon Bedrock 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, depending on the work that you do in Amazon Bedrock.

**Service user** – If you use the Amazon Bedrock service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Bedrock features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Bedrock, see Troubleshooting Amazon Bedrock identity and access.

**Service administrator** – If you're in charge of Amazon Bedrock resources at your company, you probably have full access to Amazon Bedrock. It's your job to determine which Amazon Bedrock features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Bedrock, see How Amazon Bedrock works with IAM.

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Bedrock. To view example Amazon Bedrock identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Bedrock.

## Authenticating with identities

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

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

### AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

## Federated identity

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

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

## IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

## IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must

  have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role –** 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 Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role –** 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.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) 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 is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

## Identity-based policies

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 Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing 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 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.

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.

## Access control lists (ACLs)

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.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. 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 Amazon Bedrock works with IAM

Before you use IAM to manage access to Amazon Bedrock, learn what IAM features are available to use with Amazon Bedrock.

### IAM features you can use with Amazon Bedrock

| IAM feature | Amazon Bedrock support |
|---|---|
| Identity-based policies | Yes |
| Resource-based policies | No |
| Policy actions | Yes |
| Policy resources | Yes |
| Policy condition keys | Yes |
| ACLs | No |
| ABAC (tags in policies) | Yes |
| Temporary credentials | Yes |
| Principal permissions | Yes |
| Service roles | Yes |
| Service-linked roles | No |

To get a high-level view of how Amazon Bedrock 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 Amazon Bedrock

| | |
|---|---|
| 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 Creating IAM 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. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. 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 Amazon Bedrock**

To view examples of Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

## Resource-based policies within Amazon Bedrock

| | |
|---|---|
| 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. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

## Policy actions for Amazon Bedrock

| | |
|---|---|
| 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. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Bedrock actions, see Actions defined by Amazon Bedrock in the *Service Authorization Reference*.

Policy actions in Amazon Bedrock use the following prefix before the action:

```
bedrock
```

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

```
"Action": [
   "bedrock:action1",
   "bedrock:action2"
]
```

To view examples of Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

## Policy resources for Amazon Bedrock

| | |
|---|---|
| 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. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

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

Some Amazon Bedrock API actions support multiple resources. For example, AssociateAgentKnowledgeBase accesses *AGENT12345* and *KB12345678*, so a principal must have permissions to access both resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
   "arn:aws:bedrock:aws-region:111122223333:agent/AGENT12345",
   "arn:aws:bedrock:aws-region:111122223333:knowledge-base/KB12345678"
]
```

To view examples of Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

## Policy condition keys for Amazon Bedrock

| | |
|---|---|
| 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 (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. 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.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

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

All Amazon Bedrock actions support condition keys using Amazon Bedrock models as the resource.

To view examples of Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

## ACLs in Amazon Bedrock

| | |
|---|---|
| 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 Amazon Bedrock

| | |
|---|---|
| Supports ABAC (tags in policies) | Yes |

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

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 What is ABAC? 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 Amazon Bedrock

| | |
|---|---|
| Supports temporary credentials | Yes |

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

## Cross-service principal permissions for Amazon Bedrock

| | |
|---|---|
| Supports forward access sessions (FAS) | Yes |

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

## Service roles for Amazon Bedrock

| | |
|---|---|
| Supports service roles | Yes |

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 Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

**Warning**

Changing the permissions for a service role might break Amazon Bedrock functionality. Edit service roles only when Amazon Bedrock provides guidance to do so.

## Service-linked roles for Amazon Bedrock

| | |
|---|---|
| Supports service-linked roles | No |

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.

## Identity-based policy examples for Amazon Bedrock

By default, users and roles don't have permission to create or modify Amazon Bedrock resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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

For details about actions and resource types defined by Amazon Bedrock, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Bedrock in the *Service Authorization Reference*.

**Topics**

### Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Bedrock 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 AWS 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 IAM Access Analyzer policy validation 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 Configuring MFA-protected API access in the *IAM User Guide*.

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

## Use the Amazon Bedrock console

To access the Amazon Bedrock console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Bedrock 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 Amazon Bedrock console, also attach the Amazon Bedrock AmazonBedrockFullAccess or AmazonBedrockReadOnly AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

## Allow 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 access to third-party model subscriptions

To access the Amazon Bedrock models for the first time, you use the Amazon Bedrock console to subscribe to third-party models. Your IAM user or role that the console user assumes requires permission to access the subscription API operations.

The following example shows an identity-based policy to allow access to the subscription API operations.

Use a condition key, as in the example, to limits the scope of the policy to a subset of the Amazon Bedrock foundation models in the Marketplace. To see a list of product IDs and which foundation models they correspond to, see the table in Control model access permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:Subscribe"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws-marketplace:ProductId": [
                        "1d288c71-65f9-489a-a3e2-9c7f4f6e6a85",
                        "cc0bdd50-279a-40d8-829c-4009b77a1fcc",
                        "c468b48a-84df-43a4-8c46-8870630108a7",
                        "99d90be8-b43e-49b7-91e4-752f3866c8c7",
                        "b0eb9475-3a2c-43d1-94d3-56756fd43737",
                        "d0123e8d-50d6-4dba-8a26-3fed4899f388",
                        "a61c46fe-1747-41aa-9af0-2e0ae8a9ce05",
                        "216b69fd-07d5-4c7b-866b-936456d68311",
                        "b7568428-a1ab-46d8-bab3-37def50f6f6a",
                        "38e55671-c3fe-4a44-9783-3584906e7cad",
                        "prod-ariujvyzvd2qy",
                        "prod-2c2yc2s3guhqy",
                        "prod-6dw3qvchef7zy",
                        "prod-ozonys2hmmpeu"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:Unsubscribe",
                "aws-marketplace:ViewSubscriptions"
            ],
            "Resource": "*"
        }
    ]
}
```

## Deny access for inference on specific models

The following example shows an identity-based policy that denies access to running inference on a specific model.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "DenyInference",
        "Effect": "Deny",
        "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ],
        "Resource": "arn:aws:bedrock:*::foundation-model/model-id"
    }
}
```

## Identity-based policy examples for Agents for Amazon Bedrock

Select a topic to see example IAM policies that you can attach to an IAM role to provision permissions for actions in Agents for Amazon Bedrock.

**Topics**

### Required permissions for Agents for Amazon Bedrock

For an IAM identity to use Agents for Amazon Bedrock, you must configure it with the necessary permissions. You can attach the AmazonBedrockFullAccess policy to grant the proper permissions to the role.

To restrict permissions to only actions that are used in Agents for Amazon Bedrock, attach the following identity-based policy to an IAM role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Agents for Amazon Bedrock permissions",
            "Effect": "Allow",
            "Action": [
                "bedrock:ListFoundationModels",
                "bedrock:GetFoundationModel",
                "bedrock:TagResource",
                "bedrock:UntagResource",
                "bedrock:ListTagsForResource",
                "bedrock:CreateAgent",
                "bedrock:UpdateAgent",
                "bedrock:GetAgent",
                "bedrock:ListAgents",
                "bedrock:DeleteAgent",
                "bedrock:CreateAgentActionGroup",
                "bedrock:UpdateAgentActionGroup",
                "bedrock:GetAgentActionGroup",
                "bedrock:ListAgentActionGroups",
                "bedrock:DeleteAgentActionGroup",
                "bedrock:GetAgentVersion",
                "bedrock:ListAgentVersions",
                "bedrock:DeleteAgentVersion",
                "bedrock:CreateAgentAlias",
                "bedrock:UpdateAgentAlias",
                "bedrock:GetAgentAlias",
                "bedrock:ListAgentAliases",
                "bedrock:DeleteAgentAlias",
                "bedrock:AssociateAgentKnowledgeBase",
                "bedrock:DisassociateAgentKnowledgeBase",
                "bedrock:GetKnowledgeBase",
                "bedrock:ListKnowledgeBases",
                "bedrock:PrepareAgent",
                "bedrock:InvokeAgent"
            ],
            "Resource": "*"
        }
    ]
}
```

You can further restrict permissions by omitting actions or specifying resources and condition keys. An IAM identity can call API operations on specific resources. For example, the UpdateAgent operation can only be used on agent resources and the InvokeAgent operation can only be used on alias resources. For API operations that aren't used on a specific resource type (such as CreateAgent), specify * as the `Resource`. If you specify an API operation that can't be used on the resource specified in the policy, Amazon Bedrock returns an error.

**Allow users to view information about and invoke an agent**

The following is a sample policy that you can attach to an IAM role to allow it to view information about or edit an agent with the ID *AGENT12345* and to interact with its alias with the ID *ALIAS12345*. For example, you could attach this policy to a role that you want to only have permissions to troubleshoot an agent and update it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Get information about and update an agent",
            "Effect": "Allow",
            "Action": [
                "bedrock:GetAgent",
                "bedrock:UpdateAgent"
            ],
            "Resource": "arn:aws:bedrock:aws-region:111122223333:agent/AGENT12345"
        },
        {
            "Sid": "Invoke an agent",
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeAgent"
            ],
            "Resource": "arn:aws:bedrock:aws-region:111122223333:agent-
alias/AGENT12345/ALIAS12345"
```

```
        },
    ]
}
```

## Identity-based policy examples for Provisioned Throughput

Select a topic to see example IAM policies that you can attach to an IAM role to provision permissions for actions related to Provisioned Throughput for Amazon Bedrock.

**Topics**

### Required permissions for Provisioned Throughput

For an IAM identity to use Provisioned Throughput, you must configure it with the necessary permissions. You can attach the AmazonBedrockFullAccess policy to grant the proper permissions to the role.

To restrict permissions to only actions that are used in Provisioned Throughput, attach the following identity-based policy to an IAM role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Provisioned Throughput permissions",
            "Effect": "Allow",
            "Action": [
                "bedrock:GetFoundationModel",
                "bedrock:ListFoundationModels",
                "bedrock:InvokeModel",
                "bedrock:InvokeModelWithResponseStream",
                "bedrock:ListTagsForResource",
                "bedrock:UntagResource",
                "bedrock:TagResource",
                "bedrock:CreateProvisionedModelThroughput",
                "bedrock:GetProvisionedModelThroughput",
                "bedrock:ListProvisionedModelThroughputs",
                "bedrock:UpdateProvisionedModelThroughput",
                "bedrock:DeleteProvisionedModelThroughput"
            ],
            "Resource": "*"
        }
    ]
}
```

You can further restrict permissions by omitting actions or specifying resources and condition keys. An IAM identity can call API operations on specific resources. For example, the CreateProvisionedModelThroughput operation can only be used on custom model and foundation model resources and the DeleteProvisionedModelThroughput operation can only be used on provisioned model resources. For API operations that aren't used on a specific resource type (such as ListProvisionedModelThroughputs), specify * as the `Resource`. If you specify an API operation that can't be used on the resource specified in the policy, Amazon Bedrock returns an error.

**Allow users to invoke a provisioned model**

The following is a sample policy that you can attach to an IAM role to allow it to use a provisioned model in model inference. For example, you could attach this policy to a role that you want to only have permissions to use a provisioned model. The role won't be able to manage or see information about the Provisioned Throughput.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Use a Provisioned Throughput for model inference",
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel",
                "bedrock:InvokeModelWithResponseStream"
            ],
            "Resource": "arn:aws:bedrock:aws-region:111122223333:provisioned-
model/${my-provisioned-model}"
        }
    ]
}
```

## AWS managed policies for Amazon Bedrock

To add permissions to users, groups, and roles, it's easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

### AWS managed policy: AmazonBedrockFullAccess

You can attach the `AmazonBedrockFullAccess` policy to your IAM identities.

This policy grants administrative permissions that allow the user permission to create, read, update, and delete Amazon Bedrock resources.

**Note**

Fine-tuning and model access require extra permissions. See Allow access to third-party model subscriptions and Permissions to access training and validation files and to write output files in S3 for more information.

#### Permissions details

This policy includes the following permissions:

- `ec2` (Amazon Elastic Compute Cloud) – Allows permissions to describe VPCs, subnets, and security groups.
- `iam` (AWS Identity and Access Management) – Allows principals to pass roles, but only allows IAM roles with "Amazon Bedrock" in them to be passed to the Amazon Bedrock service. The permissions are restricted to `bedrock.amazonaws.com` for Amazon Bedrock operations.
- `kms` (AWS Key Management Service) – Allows principals to describe AWS KMS keys and aliases.
- `bedrock` (Amazon Bedrock) – Allows principals read and write access to all actions in the Amazon Bedrock control plane and runtime service.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BedrockAll",
            "Effect": "Allow",
            "Action": [
                "bedrock:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DescribeKey",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:*:kms:*:::*"
        },
        {
            "Sid": "APIsWithAllResourceAccess",
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassRoleToBedrock",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/*AmazonBedrock*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "bedrock.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

## AWS managed policy: AmazonBedrockReadOnly

You can attach the `AmazonBedrockReadOnly` policy to your IAM identities.

This policy grants read-only permissions that allow users to view all resources in Amazon Bedrock.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonBedrockReadOnly",
            "Effect": "Allow",
            "Action": [
                "bedrock:GetFoundationModel",
                "bedrock:ListFoundationModels",
                "bedrock:GetModelInvocationLoggingConfiguration",
                "bedrock:GetProvisionedModelThroughput",
                "bedrock:ListProvisionedModelThroughputs",
                "bedrock:GetModelCustomizationJob",
                "bedrock:ListModelCustomizationJobs",
                "bedrock:ListCustomModels",
                "bedrock:GetCustomModel",
                "bedrock:ListTagsForResource",
                "bedrock:GetFoundationModelAvailability"
            ],
            "Resource": "*"
        }
    ]
}
```

## Amazon Bedrock updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Bedrock since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Document history for the Amazon Bedrock User Guide.

| Change | Description | Date |
|---|---|---|
| AmazonBedrockFullAccess – New policy | Amazon Bedrock added a new policy to give users permissions to create, read, update, and delete resources. | December 12, 2023 |
| AmazonBedrockReadOnly – New policy | Amazon Bedrock added a new policy to give users read-only permissions for all actions. | December 12, 2023 |
| Amazon Bedrock started tracking changes | Amazon Bedrock started tracking changes for its AWS managed policies. | December 12, 2023 |

## Service roles

Amazon Bedrock uses IAM service roles for the following features to let Amazon Bedrock carry out tasks on your behalf.

The console automatically creates service roles for supported features.

You can also create a custom service role and customize the attached permissions to your specific use-case. If you use the console, you can select this role instead of letting Amazon Bedrock create one for you.

To set up the custom service role, you carry out the following general steps.

1. Create the role by following the steps at Creating a role to delegate permissions to an AWS service.
2. Attach a **trust policy**.
3. Attach the relevant **identity-based permissions**.

Refer to the following links for more information about IAM concepts that are relevant to setting service role permissions.

- AWS service role
- Identity-based policies and resource-based policies
- Using resource-based policies for Lambda
- AWS global condition context keys
- Condition keys for Amazon Bedrock

Select a topic to learn more about service roles for a specific feature.

**Topics**

## Create a service role for model customization

To use a custom role for model customization instead of the one Amazon Bedrock automatically creates, create an IAM role and attach the following permissions by following the steps at Creating a role to delegate permissions to an AWS service.

- Trust relationship
- Permissions to access your training and validation data in S3 and to write your output data to S3
- (Optional) If you encrypt any of the following resources with a KMS key, permissions to decrypt the key (see Encryption of model customization jobs and artifacts)
  - A model customization job or the resulting custom model
  - The training, validation, or output data for the model customization job

**Topics**

- Trust relationship
- Permissions to access training and validation files and to write output files in S3

### Trust relationship

The following policy allows Amazon Bedrock to assume this role and carry out the model customization job. The following shows an example policy you can use.

You can optionally restrict the scope of the permission for cross-service confused deputy prevention by using one or more global condition context keys with the `Condition` field. For more information, see AWS global condition context keys.

- Set the `aws:SourceAccount` value to your account ID.
- (Optional) Use the `ArnEquals` or `ArnLike` condition to restrict the scope to specific model customization jobs in your account ID.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account-id"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:account-id:model-
customization-job/*"
                }
            }
        }
    ]
}
```

### Permissions to access training and validation files and to write output files in S3

Attach the following policy to allow the role to access your training and validation data and the bucket to which to write your output data. Replace the values in the `Resource` list with your actual bucket names.

To restrict access to a specific folder in a bucket, add an `s3:prefix` condition key with your folder path. You can follow the **User policy** example in Example 2: Getting a list of objects in a bucket with a specific prefix

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::training-bucket",
                "arn:aws:s3:::training-bucket/*",
                "arn:aws:s3:::validation-bucket",
                "arn:aws:s3:::validation-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::output-bucket",
                "arn:aws:s3:::output-bucket/*"
            ]
        }
    ]
}
```

## Create a service role for Agents for Amazon Bedrock

To use a custom service role for agents instead of the one Amazon Bedrock automatically creates, create an IAM role with the prefix `AmazonBedrockExecutionRoleForAgents_` and attach the following permissions by following the steps at Creating a role to delegate permissions to an AWS service.

- Trust policy
- A policy containing the following identity-based permissions
  - Access to the Amazon Bedrock base models
  - Access to the Amazon S3 objects containing the OpenAPI schemas for the action groups in your agents
  - Permissions for Amazon Bedrock to query knowledge bases that you want to attach to your agents
  - (Optional) If you encrypt your agent with a KMS key, permissions to decrypt the key (see Encryption of agent resources)

Whether you use a custom role or not, you also need to attach a **resource-based policy** to the Lambda functions for the action groups in your agents to provide permissions for the service role to access the functions. For more information, see Resource-based policy to allow Amazon Bedrock to invoke an action group Lambda function.

**Topics**

### Trust relationship

The following trust policy allows Amazon Bedrock to assume this role and create and manage agents. Replace the *`values`* as necessary. The policy contains optional condition keys (see Condition keys for Amazon Bedrock and AWS global condition context keys) in the `Condition` field that we recommend you use as a security best practice.

**Note**

As a best practice for security purposes, replace the *** with specific agent IDs after you have created them.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:agent/*"
            }
        }
    }]
}
```

**Identity-based permissions for the Agents service role.**

Attach the following policy to provide permissions for the service role, replacing *values* as necessary. The policy contains the following statements. Omit a statement if it isn't applicable to your use-case. The policy contains optional condition keys (see Condition keys for Amazon Bedrock and AWS global condition context keys) in the `Condition` field that we recommend you use as a security best practice.

**Note**

If you encrypt your agent with a customer-managed KMS key, refer to Encryption of agent resources for further permissions you need to add.

- Permissions to use Amazon Bedrock foundation models to run model inference on prompts used in your agent's orchestration.
- Permissions to access your agent's action group API schemas in Amazon S3. Omit this statement if your agent has no action groups.
- Permissions to access knowledge bases associated with your agent. Omit this statement if your agent has no associated knowledge bases.
- Permissions to access a third-party (Pinecone or Redis Enterprise Cloud) knowledge base associated with your agent. Omit this statement if your knowledge base is first-party (Amazon OpenSearch Serverless or Amazon Aurora) or if your agent has no associated knowledge bases.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow model invocation for orchestration",
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:region::foundation-model/anthropic.claude-v2",
                "arn:aws:bedrock:region::foundation-model/anthropic.claude-v2:1",
                "arn:aws:bedrock:region::foundation-model/anthropic.claude-instant-v1"
            ]
        },
        {
            "Sid": "Allow access to action group API schemas in S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket/path/to/schema"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "account-id"
                }
            }
        },
        {
            "Sid": "Query associated knowledge bases",
            "Effect": "Allow",
            "Action": [
                "bedrock:Retrieve",
                "bedrock:RetrieveAndGenerate"
            ],
            "Resource": [
                "arn:aws:bedrock:region:account-id:knowledge-base/knowledge-base-id"
            ]
        },
        {
            "Sid": "Associate a third-party knowledge base with your agent",
            "Effect": "Allow",
            "Action": [
                "bedrock:AssociateThirdPartyKnowledgeBase",
            ],
            "Resource": "arn:aws:bedrock:region:account-id:knowledge-base/knowledge-
base-id",
            "Condition": {
                "StringEquals" : {
                    "bedrock:ThirdPartyKnowledgeBaseCredentialsSecretArn":
 "arn:aws:kms:region:account-id:key/key-id"
                }
            }
        }
    ]
}
```

### Resource-based policy to allow Amazon Bedrock to invoke an action group Lambda function

Follow the steps at Using resource-based policies for Lambda and attach the following resource-based policy to a Lambda function to allow Amazon Bedrock to access the Lambda function for your agent's action groups, replacing the *values* as necessary. The policy contains optional condition keys (see Condition keys for Amazon Bedrock and AWS global condition context keys) in the `Condition` field that we recommend you use as a security best practice.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Allow Amazon Bedrock to access action group Lambda function",
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource":  "arn:aws:lambda:region:account-id:function:function-name",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:agent/agent-id"
            }
        }
    }]
}
```

## Create a service role for Knowledge bases for Amazon Bedrock

To use a custom role for knowledge base instead of the one Amazon Bedrock automatically creates, create an IAM role and attach the following permissions by following the steps at Creating a role to delegate permissions to an AWS service. You can use the same role across your knowledge bases.

- Trust relationship
- Access to the Amazon Bedrock base models
- Access to the Amazon S3 objects containing your data sources
- (If you create a vector database in Amazon OpenSearch Service) Access to your OpenSearch Service collection
- (If you create a vector database in Amazon Aurora)
- (If you create a vector database in Pinecone or Redis Enterprise Cloud) Permissions for AWS Secrets Manager to authenticate your Pinecone or Redis Enterprise Cloud account
- (Optional) If you encrypt any of the following resources with a KMS key, permissions to decrypt the key (see Encryption of knowledge base resources).
  - Your knowledge base
  - Data sources for your knowledge base
  - Your vector database in Amazon OpenSearch Service
  - The secret for your third-party vector database in AWS Secrets Manager
  - A data ingestion job

**Topics**

### Trust relationship

The following policy allows Amazon Bedrock to assume this role and create and manage knowledge bases. The following shows an example policy you can use. You can restrict the scope of the permission by using one or more global condition context keys. For more information, see AWS global condition context keys. Set the `aws:SourceAccount` value to your account ID. Use the `ArnEquals` or `ArnLike` condition to restrict the scope to specific knowledge bases.

**Note**

As a best practice for security purposes, replace the * with specific knowledge base IDs after you have created them.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:knowledge-base/*"
            }
        }
    }]
}
```

### Permissions to access Amazon Bedrock models

Attach the following policy to provide permissions for the role to use Amazon Bedrock models to embed your source data.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:ListFoundationModels",
                "bedrock:ListCustomModels"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:region::foundation-model/amazon.titan-embed-text-v1",
                "arn:aws:bedrock:region::foundation-model/cohere.embed-english-v3",
                "arn:aws:bedrock:region::foundation-model/cohere.embed-multilingual-v3"
            ]
        }
    ]
}
```

### Permissions to access your data sources in Amazon S3

Attach the following policy to provide permissions for the role to access the Amazon S3 URIs containing the data source files for your knowledge base. In the `Resource` field, provide an Amazon S3 object containing the data sources or add the URI of each data source to the list.

If you encrypted these data sources with a AWS KMS key, attach permissions to decrypt the key to the role by following the steps at Permissions to decrypt your AWS KMS key for your data sources in Amazon S3.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::bucket/path/to/folder",
            "arn:aws:s3:::bucket/path/to/folder/*"
        ],
        "Condition": {
            "StringEquals": {
                "aws:PrincipalAccount": "account-id"
            }
        }
    }]
}
```

### (Optional) Permissions to access your vector database in Amazon OpenSearch Service

If you created a vector database in Amazon OpenSearch Service for your knowledge base, attach the following policy to your Knowledge bases for Amazon Bedrock service role to allow access to the collection. Replace *region* and *account-id* with the region and account ID to which the database belongs. Input the ID of your Amazon OpenSearch Service collection in *collection-id*. You can allow access to multiple collections by adding them to the `Resource` list.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "aoss:APIAccessAll"
        ],
        "Resource": [
            "arn:aws:aoss:region:account-id:collection/collection-id"
        ]
    }]
}
```

### (Optional) Permissions to access your Amazon Aurora database cluster

If you created a database (DB) cluster in Amazon Aurora for your knowledge base, attach the following policy to your Knowledge bases for Amazon Bedrock service role to allow access to the DB cluster and to provide read and write permissions on it. Replace *`region`* and *`account-id`* with the region and account ID to which the DB cluster belongs. Input the ID of your Amazon Aurora database cluster in *`db-cluster-id`*. You can allow access to multiple DB clusters by adding them to the `Resource` list.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Sid": "RdsDescribeStatementID",
        "Effect": "Allow",
        "Action": [
            "rds:DescribeDBClusters"
        ],
        "Resource": [
            "arn:aws:rds:region:account-id:cluster:db-cluster-id"
        ]
    },
    {
        "Sid": "DataAPIStatementID",
        "Effect": "Allow",
        "Action": [
            "rds-data:BatchExecuteStatement",
            "rds-data:ExecuteStatement"
        ],
        "Resource": [
            "arn:aws:rds:region:account-id:cluster:db-cluster-id"
        ]
    }]
}
```

### (Optional) Permissions to access a vector database configured with an AWS Secrets Manager secret

If your vector database is configured with an AWS Secrets Manager secret, attach the following policy to your Knowledge bases for Amazon Bedrock service role to allow AWS Secrets Manager to authenticate your account to access the database. Replace *region* and *account-id* with the region and account ID to which the database belongs. Replace *secret-id* with the ID of your secret.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue"
        ],
        "Resource": [
            "arn:aws:secretsmanager:region:account-id:secret:secret-id"
        ]
    }]
}
```

If you encrypted your secret with a AWS KMS key, attach permissions to decrypt the key to the role by following the steps at Permissions to decrypt an AWS Secrets Manager secret for the vector store containing your knowledge base.

### (Optional) Permissions for AWS to manage a AWS KMS key for transient data storage during data ingestion

To allow the creation of a AWS KMS key for transient data storage in the process of ingesting your data source, attach the following policy to your Knowledge bases for Amazon Bedrock service role. Replace the *region*, *account-id*, and *key-id* with the appropriate values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "kms:GenerateDataKey",
              "kms:Decrypt"
          ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
            ]
        }
    ]
}
```

## Troubleshooting Amazon Bedrock identity and access

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

**Topics**

### I am not authorized to perform an action in Amazon Bedrock

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 `bedrock:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 bedrock: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 `bedrock:`*`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 Amazon Bedrock.

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 Amazon Bedrock. 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 Amazon Bedrock 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 Amazon Bedrock supports these features, see How Amazon Bedrock 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 How IAM roles differ from resource-based policies in the *IAM User Guide*.

## Compliance validation for Amazon Bedrock

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. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your

compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

## Incident response in Amazon Bedrock

Security is the highest priority at AWS. As part of the AWS Cloud shared responsibility model, AWS manages a data center, network, and software architecture that meets the requirements of the most security-sensitive organizations. AWS is responsible for any incident response with respect to the Amazon Bedrock service itself. Also, as an AWS customer, you share a responsibility for maintaining security in the cloud. This means that you control the security you choose to implement from the AWS tools and features you have access to. In addition, you’re responsible for incident response on your side of the shared responsibility model.

By establishing a security baseline that meets the objectives for your applications running in the cloud, you're able to detect deviations that you can respond to. To help you understand the impact that incident response and your choices have on your corporate goals, we encourage you to review the following resources:

- AWS Security Incident Response Guide
- AWS Best Practices for Security, Identity, and Compliance
- Security Perspective of the AWS Cloud Adoption Framework (CAF) whitepaper

## Resilience in Amazon Bedrock

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 zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

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

## Infrastructure security in Amazon Bedrock

As a managed service, Amazon Bedrock is protected by the 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 Amazon Bedrock 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.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

## Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon Bedrock gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:bedrock:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be ResourceDescription.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Bedrock to prevent the confused deputy problem.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
                }
            }
        }
    ]
}
```

## Configuration and vulnerability analysis in Amazon Bedrock

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

# Use interface VPC endpoints (AWS PrivateLink)

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

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 Amazon Bedrock.

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

## Considerations for Amazon Bedrock VPC endpoints

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

Amazon Bedrock supports making the following API calls through VPC endpoints.

| Category | Endpoint prefix |
|---|---|
| Amazon Bedrock Control Plane API actions | `bedrock` |
| Amazon Bedrock Runtime API actions | `bedrock-runtime` |
| Agents for Amazon Bedrock Build-time API actions | `bedrock-agent` |
| Agents for Amazon Bedrock Runtime API actions | `bedrock-agent-runtime` |

### Availability Zones

Amazon Bedrock and Agents for Amazon Bedrock endpoints are available in multiple Availability Zones.

## Create an interface endpoint for Amazon Bedrock

You can create an interface endpoint for Amazon Bedrock 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 Amazon Bedrock using any of the following service names:

- `com.amazonaws.region.bedrock`
- `com.amazonaws.region.bedrock-runtime`
- `com.amazonaws.region.bedrock-agent`
- `com.amazonaws.region.bedrock-agent-runtime`

After you create the endpoint, you have the option to enable a private DNS hostname. Enable this setting by selecting Enable Private DNS Name in the VPC console when you create the VPC endpoint.

If you enable private DNS for the interface endpoint, you can make API requests to Amazon Bedrock using its default Regional DNS name. The following examples show the format of the default Regional DNS names.

- `bedrock.region.amazonaws.com`
- `bedrock-runtime.region.amazonaws.com`
- `bedrock-agent.region.amazonaws.com`
- `bedrock-agent-runtime.region.amazonaws.com`

## Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Amazon Bedrock through the interface endpoint. To control the access allowed to Amazon Bedrock from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.

- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

**Example: VPC endpoint policy for Amazon Bedrock actions**

The following is an example of a custom endpoint policy. When you attach this resource-based policy to your interface endpoint, it grants access to the listed Amazon Bedrock actions for all principals on all resources.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ],
         "Resource":"*"
      }
   ]
}
```