Policies and permissions in AWS Identity and Access Management

Archived: 2026-04-05 19:25:18 UTC

Manage access in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles) or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. AWS supports seven types of policies: identity-based policies, resource-based policies, permissions boundaries, AWS Organizations service control policies (SCPs), AWS Organizations resource control policies (RCPs), access control lists (ACLs), and session policies.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, if a policy allows the GetUser action, then a user with that policy can get user information from the AWS Management Console, the AWS CLI, or the AWS API. When you create an IAM user, you can choose to allow console or programmatic access. If console access is allowed, the IAM user can sign in to the console using their sign-in credentials. If programmatic access is allowed, the user can use access keys to work with the CLI or API.

Policy types

The following policy types, listed in order from most frequently used to less frequently used, are available for use in AWS. For more details, see the sections below for each policy type.

- **Identity-based policies** – Attach managed and inline policies to IAM identities (users, groups to which users belong, or roles). Identity-based policies grant permissions to an identity.
- **Resource-based policies** – Attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies. Resource-based policies grant permissions to the principal that is specified in the policy. Principals can be in the same account as the resource or in other accounts.
- **Permissions boundaries** – Use a managed policy as the permissions boundary for an IAM entity (user or role). That policy defines the maximum permissions that the identity-based policies can grant to an entity, but does not grant permissions. Permissions boundaries do not define the maximum permissions that a resource-based policy can grant to an entity.
- **AWS Organizations SCPs** – Use an AWS Organizations service control policy (SCP) to define the maximum permissions for IAM users and IAM roles within accounts in your organization or organizational unit (OU). SCPs limit permissions that identity-based policies or resource-based policies grant to IAM users or IAM roles within the account. SCPs do not grant permissions.
- **AWS Organizations RCPs** – Use an AWS Organizations resource control policy (RCP) to define the maximum permissions for resources within accounts in your organization or organizational unit (OU).

RCPs limit permissions that identity-based and resource-based policies can grant to resources in accounts within your organization. RCPs do not grant permissions.

- **Access control lists (ACLs)** – Use ACLs to control which principals in other accounts can access the resource to which the ACL is attached. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document structure. ACLs are cross-account permissions policies that grant permissions to the specified principal. ACLs cannot grant permissions to entities within the same account.

- **Session policies** – Pass advanced session policies when you use the AWS CLI or AWS API to assume a role or a federated user. Session policies limit the permissions that the role or user's identity-based policies grant to the session. Session policies limit permissions for a created session, but do not grant permissions. For more information, see Session Policies.

Identity-based policies

Identity-based policies are JSON permissions policy documents that control what actions an identity (users, groups of users, and roles) can perform, on which resources, and under what conditions. Identity-based policies can be further categorized:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. There are two types of managed policies:
    - **AWS managed policies** – Managed policies that are created and managed by AWS.
    - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies.
- **Inline policies** – Policies that you add directly to a single user, group, or role. Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity.

To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. These policies grant the specified principal permission to perform specific actions on that resource and defines under what conditions this applies. Resource-based policies are inline policies. There are no managed resource-based policies.

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 separate AWS accounts, you must

also use an identity-based policy to grant the principal access to the resource. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For step-by step instructions for granting cross-service access, see IAM tutorial: Delegate access across AWS accounts using IAM roles.

The IAM service supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. An IAM role is both an identity and a resource that supports resource-based policies. For that reason, you must attach both a trust policy and an identity-based policy to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. To learn how IAM roles are different from other resource-based policies, see Cross account resource access in IAM.

To see which other services support resource-based policies, see AWS services that work with IAM. To learn more about resource-based policies, see Identity-based policies and resource-based policies. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

IAM 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. When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. If you specify a role session or user in the principal element of a resource-based policy, an explicit allow in the permission boundary is not required. However, if you specify a role ARN in the principal element of a resource-based policy, an explicit allow in the permission boundary is required. In both cases, an explicit deny in the permission boundary is effective. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities.

AWS Organizations service control policies (SCPs)

If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. SCPs are JSON policies that specify the maximum permissions for IAM users and IAM roles within accounts of an organization or organizational unit (OU). The SCP limits permissions for principals in member accounts, including each AWS account root user. An explicit deny in any of these policies overrides an allow in other policies.

For more information about AWS Organizations and SCPs, see Service control policies (SCPs) in the *AWS Organizations User Guide*.

AWS Organizations resource control policies (RCPs)

If you enable all features in an organization, then you can use resource control policies (RCPs) to centrally apply access controls on resources across multiple AWS accounts. RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to

your organization. An explicit deny in any applicable RCP overrides an allow in other policies that might be attached to individual identities or resources.

For more information about AWS Organizations and RCPs including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

Access control lists (ACLs)

Access control lists (ACLs) are service policies that allow you to control which principals in another account can access a resource. ACLs cannot be used to control access for a principal within the same account. ACLs are similar to resource-based policies, although they are the only policy type that does 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*.

Session policies

Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or an AWS STS federated user principal. The permissions for a session are the intersection of the identity-based policies for the IAM entity (user or role) used to create the session and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow.

You can create role session and pass session policies programmatically using the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. For more information about creating a role session, see Permissions for temporary security credentials.

When you create an AWS STS federated user principal session, you use the access keys of the IAM user to programmatically call the `GetFederationToken` API operation. You must also pass session policies. The resulting session's permissions are the intersection of the identity-based policy and the session policy. For more information about creating a federated user session, see Requesting credentials through a custom identity broker.

A resource-based policy can specify the ARN of the user or role as a principal. In that case, the permissions from the resource-based policy are added to the role or user's identity-based policy before the session is created. The session policy limits the total permissions granted by the resource-based policy and the identity-based policy. The resulting session's permissions are the intersection of the session policies and the resource-based policies plus the intersection of the session policies and identity-based policies.

A resource-based policy can specify the ARN of the session as a principal. In that case, the permissions from the resource-based policy are added after the session is created. The resource-based policy permissions are not limited by the session policy. The resulting session has all the permissions of the resource-based policy *plus* the intersection of the identity-based policy and the session policy.

A permissions boundary can set the maximum permissions for a user or role that is used to create a session. In that case, the resulting session's permissions are the intersection of the session policy, the permissions boundary, and the identity-based policy. However, a permissions boundary does not limit permissions granted by a resource-based policy that specifies the ARN of the resulting session.

Policies and the root user

The AWS account root user is affected by some policy types but not others. You cannot attach identity-based policies to the root user, and you cannot set the permissions boundary for the root user. However, you can specify the root user as the principal in a resource-based policy or an ACL. A root user is still the member of an account. If that account is a member of an organization in AWS Organizations, the root user is affected by SCPs and RCPs for the account.

Overview of JSON policies

Most policies are stored in AWS as JSON documents. Identity-based policies and policies used to set permissions boundaries are JSON policy documents that you attach to a user or role. Resource-based policies are JSON policy documents that you attach to a resource. SCPs and RCPs are JSON policy documents with restricted syntax that you attach to the AWS Organizations' organization root, organizational unit (OU), or an account. ACLs are also attached to a resource, but you must use a different syntax. Session policies are JSON policies that you provide when you assume a role or federated user session.

It is not necessary for you to understand the JSON syntax. You can use the visual editor in the AWS Management Console to create and edit customer managed policies without ever using JSON. However, if you use inline policies for groups or complex policies, you must still create and edit those policies in the JSON editor using the console. For more information about using the visual editor, see Define custom IAM permissions with customer managed policies and Edit IAM policies.

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

JSON policy document structure

As illustrated in the following figure, a JSON policy document includes these elements:

- Optional policy-wide information at the top of the document
- One or more individual statements

Each statement includes information about a single permission. If a policy includes multiple statements, AWS applies a logical `OR` across the statements when evaluating them. If multiple policies apply to a request, AWS applies a logical `OR` across all of those policies when evaluating them.

The information in a statement is contained within a series of elements.

- **Version** – Specify the version of the policy language that you want to use. We recommend that you use the latest `2012-10-17` version. For more information, see IAM JSON policy elements: Version
- **Statement** – Use this main policy element as a container for the following elements. You can include more than one statement in a policy.
- **Sid** (Optional) – Include an optional statement ID to differentiate between your statements.
- **Effect** – Use `Allow` or `Deny` to indicate whether the policy allows or denies access.
- **Principal** (Required in some circumstances) – If you create a resource-based policy, you must indicate the account, user, role, or AWS STS federated user principal to which you would like to allow or deny access. If you are creating an IAM permissions policy to attach to a user or role, you cannot include this element. The principal is implied as that user or role.
- **Action** – Include a list of actions that the policy allows or denies.
- **Resource** (Required in some circumstances) – If you create an IAM permissions policy, you must specify a list of resources to which the actions apply. If you create a resource-based policy, it depends on the resource you're using as to whether this element is required or not.
- **Condition** (Optional) – Specify the circumstances under which the policy grants permission.

To learn about these and other more advanced policy elements, see IAM JSON policy element reference.

Multiple statements and multiple policies

If you want to define more than one permission for an entity (user or role), you can use multiple statements in a single policy. You can also attach multiple policies. If you try to define multiple permissions in a single statement, your policy might not grant the access that you expect. We recommend that you break up policies by resource type.

Because of the limited size of policies, it might be necessary to use multiple policies for more complex permissions. It's also a good idea to create functional groupings of permissions in a separate customer managed policy. For example, Create one policy for IAM user management, one for self-management, and another policy for S3 bucket management. Regardless of the combination of multiple statements and multiple policies, AWS evaluates your policies the same way.

For example, the following policy has three statements, each of which defines a separate set of permissions within a single account. The statements define the following:

- The first statement, with an `Sid` (Statement ID) of `FirstStatement`, lets the user with the attached policy change their own password. The `Resource` element in this statement is "`*`" (which means "all resources"). But in practice, the `ChangePassword` API operation (or equivalent `change-password` CLI command) affects only the password for the user who makes the request.

- The second statement lets the user list all the Amazon S3 buckets in their AWS account. The `Resource` element in this statement is `"*"` (which means "all resources"). But because policies don't grant access to resources in other accounts, the user can list only the buckets in their own AWS account.
- The third statement lets the user list and retrieve any object that is in a bucket named `amzn-s3-demo-bucket-confidential-data`, but only when the user is authenticated with multi-factor authentication (MFA). The `Condition` element in the policy enforces the MFA authentication.

  When a policy statement contains a `Condition` element, the statement is only in effect when the `Condition` element evaluates to true. In this case, the `Condition` evaluates to true when the user is MFA-authenticated. If the user is not MFA-authenticated, this `Condition` evaluates to false. In that case, the third statement in this policy does not apply and the user does not have access to the `amzn-s3-demo-bucket-confidential-data` bucket.

```
  {
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "FirstStatement",
      "Effect": "Allow",
      "Action": ["iam:ChangePassword"],
      "Resource": "*"
    },
    {
      "Sid": "SecondStatement",
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    },
    {
      "Sid": "ThirdStatement",
      "Effect": "Allow",
      "Action": [
        "s3:List*",
        "s3:Get*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket-confidential-data",
        "arn:aws:s3:::amzn-s3-demo-bucket-confidential-data/*"
      ],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

Examples of JSON policy syntax

The following identity-based policy allows the implied principal to list a single Amazon S3 bucket named `amzn-s3-demo-bucket`:

```
  {
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
  }
}
```

The following resource-based policy can be attached to an Amazon S3 bucket. The policy allows members of a specific AWS account to perform any Amazon S3 actions in the bucket named `amzn-s3-demo-bucket`. It allows any action that can be performed on a bucket or the objects within it. (Because the policy grants trust only to the account, individual users in the account must still be granted permissions for the specified Amazon S3 actions.)

```
  {
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam:: 111122223333 :root"
                ]
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

To view example policies for common scenarios, see Example IAM identity-based policies.

Grant least privilege

When you create IAM policies, follow the standard security advice of granting *least privilege*, or granting only the permissions required to perform a task. Determine what users and roles need to do and then craft policies that allow them to perform *only* those tasks.

Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later.

As an alternative to least privilege, you can use AWS managed policies or policies with wildcard `*` permissions to get started with policies. Consider the security risk of granting your principals more permissions than they need to do their job. Monitor those principals to learn which permissions they are using. Then write least privilege policies.

IAM provides several options to help you refine the permissions that you grant.

- **Understand access level groupings** – You can use access level groupings to understand the level of access that a policy grants. Policy actions are classified as `List`, `Read`, `Write`, `Permissions management`, or `Tagging`. For example, you can choose actions from the `List` and `Read` access levels to grant read-only access to your users. To learn how to use policy summaries to understand access level permissions, see Access levels in policy summaries.

- **Validate your policies** – You can perform policy validation using IAM Access Analyzer when you create and edit JSON policies. We recommend that you review and validate all of your existing policies. IAM Access Analyzer provides over 100 policy checks to validate your policies. It generates security warnings when a statement in your policy allows access we consider overly permissive. You can use the actionable recommendations that are provided through the security warnings as you work toward granting least privilege. To learn more about policy checks provided by IAM Access Analyzer, see IAM Access Analyzer policy validation.

- **Generate a policy based on access activity** – To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an IAM entity (user or role). IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified time frame. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see IAM Access Analyzer policy generation.

- **Use last accessed information** – Another feature that can help with least privilege is *last accessed information*. View this information on the **Access Advisor** tab on the IAM console details page for an IAM user, group, role, or policy. Last accessed information also includes information about the actions that were last accessed for some services, such as Amazon EC2, IAM, Lambda, and Amazon S3. If you sign in using AWS Organizations management account credentials, you can view service last accessed information in the **AWS Organizations** section of the IAM console. You can also use the AWS CLI or AWS API to retrieve a report for last accessed information for entities or policies in IAM or AWS Organizations. You can use this information to identify unnecessary permissions so that you can refine your IAM or AWS Organizations policies to better adhere to the principle of least privilege. For more information, see Refine permissions in AWS using last accessed information.

- **Review account events in AWS CloudTrail** – To further reduce permissions, you can view your account's events in AWS CloudTrail **Event history**. CloudTrail event logs include detailed event information that

you can use to reduce the policy's permissions. The logs include only the actions and resources that your IAM entities need. For more information, see Viewing CloudTrail Events in the CloudTrail Console in the *AWS CloudTrail User Guide*.

For more information, see the following policy topics for individual services, which provide examples of how to write policies for service-specific resources.

- Using resource-based policies for DynamoDB in the *Amazon DynamoDB Developer Guide*
- Bucket policies for Amazon S3 in the *Amazon Simple Storage Service User Guide*
- Access Control List (ACL) overview in the *Amazon Simple Storage Service User Guide*
