AWS Security Blog

# IAM policy types: How and when to use them

by Matt Luttrell and Josh Joy | on 03 JUN 2022 | in Intermediate (200), Security, Identity, & Compliance | Permalink | Comments | Share

You manage access in AWS by creating policies and attaching them to AWS Identity and Access Management (IAM) principals (roles, users, or groups of users) or AWS resources. AWS evaluates these policies when an IAM principal makes a request, such as uploading an object to an Amazon Simple Storage Service (Amazon S3) bucket. Permissions in the policies determine whether the request is allowed or denied.

In this blog post, we will walk you through a scenario and explain when you should use which policy type, and who should own and manage the policy. You will learn when to use the more common policy types: identity-based policies, resource-based policies, permissions boundaries, and AWS Organizations service control policies (SCPs).

## Different policy types and when to use them

AWS has different policy types that provide you with powerful flexibility, and it's important to know how and when to use each policy type. It's also important for you to understand how to structure your IAM policy ownership to avoid a centralized team from becoming a bottleneck. Explicit policy ownership can allow your teams to move more quickly, while staying within the secure guardrails that are defined centrally.

### Service control policies overview

Service control policies (SCPs) are a feature of AWS Organizations. AWS Organizations is a service for grouping and centrally managing the AWS accounts that your business owns. SCPs are policies that specify the maximum permissions for an organization, organizational unit (OU), or an individual account. An SCP can limit permissions for principals in member accounts, including the AWS account root user.

SCPs are meant to be used as coarse-grained guardrails, and they don't directly grant access. The primary function of SCPs is to enforce *security invariants* across AWS accounts and OUs in an organization. *Security invariants* are control objectives or configurations that you apply to multiple accounts, OUs, or the whole AWS organization. For example, you can use an SCP to prevent member accounts from leaving your organization or to enforce that AWS resources can only be deployed to certain Regions.

### Permissions boundaries overview

Permissions boundaries are an advanced IAM feature in which you set the maximum permissions that an identity-based policy can grant to an IAM principal. When you set a permissions boundary for a principal, the principal can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

A permissions boundary is a type of identity-based policy that doesn't directly grant access. Instead, like an SCP, a permissions boundary acts as a guardrail for your IAM principals that allows you to set coarse-grained access controls. A permissions boundary is typically used to delegate the creation of IAM principals. Delegation enables other individuals in your accounts to create new IAM principals, but limits the permissions that can be granted to the new IAM principals.

### Identity-based policies overview

Identity-based policies are policy documents that you attach to a principal (roles, users, and groups of users) to control what actions a principal can perform, on which resources, and under what conditions. Identity-based policies can be further categorized into AWS managed policies, customer managed policies, and inline policies. AWS managed policies are reusable

identity-based policies that are created and managed by AWS. You can use AWS managed policies as a starting point for building your own identity-based policies that are specific to your organization. Customer managed policies are reusable identity-based policies that can be attached to multiple identities. Customer managed policies are useful when you have multiple principals with identical access requirements. Inline policies are identity-based policies that are attached to a single principal. Use inline-policies when you want to create least-privilege permissions that are specific to a particular principal.

You will have many identity-based policies in your AWS account that are used to enable access in scenarios such as human access, application access, machine learning workloads, and deployment pipelines. These policies should be fine-grained. You use these policies to directly apply least privilege permissions to your IAM principals. You should write the policies with permissions for the specific task that the principal needs to accomplish.

### Resource-based policies overview

Resource-based policies are policy documents that you attach to a resource such as an S3 bucket. These policies grant the specified principal permission to perform specific actions on that resource and define under what conditions this permission applies. Resource-based policies are inline policies. For a list of AWS services that support resource-based policies, see AWS services that work with IAM.

Resource-based policies are optional for many workloads that don't span multiple AWS accounts. Fine-grained access within a single AWS account is typically granted with identity-based policies. AWS Key Management Service (AWS KMS) keys and IAM role trust policies are two exceptions, and both of these resources must have a resource-based policy even when the principal and the KMS key or IAM role are in the same account. IAM roles and KMS keys behave this way as an extra layer of protection that requires the owner of the resource (key or role) to explicitly allow or deny principals from using the resource. For other resources that support resource-based policies, here are some use cases where they are most commonly used:

1. Granting cross-account access to your AWS resource.
2. Granting an AWS service access to your resource when the AWS service uses an AWS service principal. For example, when using AWS CloudTrail, you must explicitly grant the CloudTrail service principal access to write files to an Amazon S3 bucket.
3. Applying broad access guardrails to your AWS resources. You can see some examples in the blog post IAM makes it easier for you to manage permissions for AWS services accessing your resources.
4. Applying an additional layer of protection for resources that store sensitive data, such as AWS Secrets Manager secrets or an S3 bucket with sensitive data. You can use a resource-based policy to deny access to IAM principals that shouldn't have access to sensitive data, even if granted access by an identity-based policy. An explicit deny in an IAM policy always overrides an allow.

## How to implement different policy types

In this section, we will walk you through an example of a design that includes all four of the policy types explained in this post.

The example that follows shows an application that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance and needs to read from and write files to an S3 bucket in the same account. The application also reads (but doesn't write) files from an S3 bucket in a different account. The company in this example, Example Corp, uses a multi-account strategy, and each application has its own AWS account. The architecture of the application is shown in Figure 1.

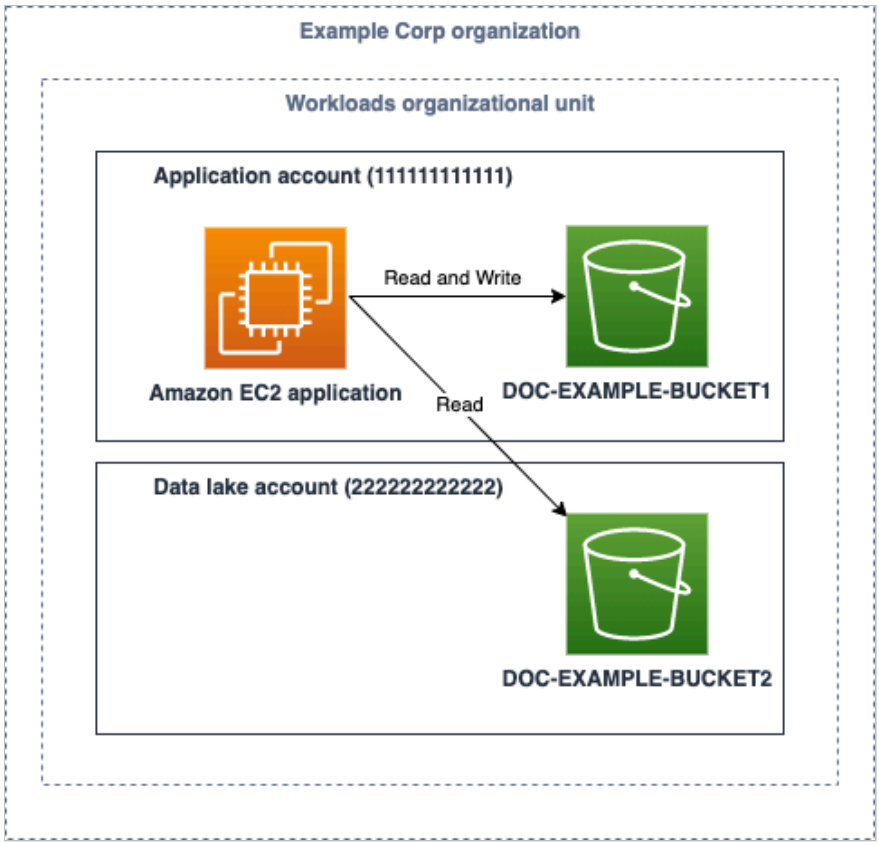


Figure 1: Sample application architecture that needs to access S3 buckets in two different AWS accounts

There are three teams that participate in this example: the Central Cloud Team, the Application Team, and the Data Lake Team. The Central Cloud Team is responsible for the overall security and governance of the AWS environment across all AWS accounts at Example Corp. The Application Team is responsible for building, deploying, and running their application within the application account (111111111111) that they own and manage. Likewise, the Data Lake Team owns and manages the data lake account (222222222222) that hosts a data lake at Example Corp.

With that background in mind, we will walk you through an implementation for each of the four policy types and include an explanation of which team we recommend own each policy. The policy owner is the team that is responsible for creating and maintaining the policy.

## Service control policies

The Central Cloud Team owns the implementation of the security controls that should apply broadly to all of Example Corp's AWS accounts. At Example Corp, the Central Cloud Team has two security requirements that they want to apply to all accounts in their organization:

1. All AWS API calls must be encrypted in transit.
2. Accounts can't leave the organization on their own.

The Central Cloud Team chooses to implement these security invariants using SCPs and applies the SCPs to the root of the organization. The first statement in Policy 1 denies all requests that are not sent using SSL (TLS). The second statement in Policy 1 prevents an account from leaving the organization.

This is only a subset of the SCP statements that Example Corp uses. Example Corp uses a deny list strategy, and there must also be an accompanying statement with an Effect of Allow at every level of the organization that isn't shown in the SCP in Policy 1.

**Policy 1: SCP attached to AWS Organizations organization root**

```
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "DenyIfRequestIsNotUsingSSL",
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "BoolIfExists": {
                "aws:SecureTransport": "false"
            }
        }
    },
    {
        "Sid": "PreventLeavingTheOrganization",
        "Effect": "Deny",
        "Action": "organizations:LeaveOrganization",
        "Resource": "*"
    }]
}
```

## Permissions boundary policies

The Central Cloud Team wants to make sure that they don't become a bottleneck for the Application Team. They want to allow the Application Team to deploy their own IAM principals and policies for their applications. The Central Cloud Team also wants to make sure that any principals created by the Application Team can only use AWS APIs that the Central Cloud Team has approved.

At Example Corp, the Application Team deploys to their production AWS environment through a continuous integration/continuous deployment (CI/CD) pipeline. The pipeline itself has broad access to create AWS resources needed to run applications, including permissions to create additional IAM roles. The Central Cloud Team implements a control that requires that all IAM roles created by the pipeline must have a permissions boundary attached. This allows the pipeline to create additional IAM roles, but limits the permissions that the newly created roles can have to what is allowed by the permissions boundary. This delegation strikes a balance for the Central Cloud Team. They can avoid becoming a bottleneck to the Application Team by allowing the Application Team to create their own IAM roles and policies, while ensuring that those IAM roles and policies are not overly privileged.

An example of the permissions boundary policy that the Central Cloud Team attaches to IAM roles created by the CI/CD pipeline is shown below. This same permissions boundary policy can be centrally managed and attached to IAM roles created by other pipelines at Example Corp. The policy describes the maximum possible permissions that additional roles created by the Application Team are allowed to have, and it limits those permissions to some Amazon S3 and Amazon Simple Queue Service (Amazon SQS) data access actions. It's common for a permissions boundary policy to include data access actions when used to delegate role creation. This is because most applications only need permissions to read and write data (for example, writing an object to an S3 bucket or reading a message from an SQS queue) and only sometimes need permission to modify infrastructure (for example, creating an S3 bucket or deleting an SQS queue). As Example Corp adopts additional AWS services, the Central Cloud Team updates this permissions boundary with actions from those services.

**Policy 2: Permissions boundary policy attached to IAM roles created by the CI/CD pipeline**

```
{
    "Id": "PermissionsBoundaryPolicy",
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "sqs:ChangeMessageVisibility",
            "sqs:DeleteMessage",
            "sqs:ReceiveMessage",
            "sqs:SendMessage",
            "sqs:PurgeQueue",
            "sqs:GetQueueUrl",
            "logs:PutLogEvents"
         ],
        "Resource": "*"
    }]
}
```

In the next section, you will learn how to enforce that this permissions boundary is attached to IAM roles created by your CI/CD pipeline.

### Identity-based policies

In this example, teams at Example Corp are only allowed to modify the production AWS environment through their CI/CD pipeline. Write access to the production environment is not allowed otherwise. To support the different personas that need to have access to an application account in Example Corp, three baseline IAM roles with identity-based policies are created in the application accounts:

- A role for the CI/CD pipeline to use to deploy application resources.
- A read-only role for the Central Cloud Team, with a process for temporary elevated access.
- A read-only role for members of the Application Team.

All three of these baseline roles are owned, managed, and deployed by the Central Cloud Team.

The Central Cloud Team is given a default read-only role (`CentralCloudTeamReadonlyRole`) that allows read access to all resources within the account. This is accomplished by attaching the AWS managed ReadOnlyAccess policy to the Central Cloud Team role. You can use the IAM console to attach the ReadOnlyAccess policy, which grants read-only access to all services. When a member of the team needs to perform an action that is not covered by this policy, they follow a temporary elevated access process to make sure that this access is valid and recorded.

A read-only role is also given to developers in the Application Team (`DeveloperReadOnlyRole`) for analysis and troubleshooting. At Example Corp, developers are allowed to have read-only access to Amazon EC2, Amazon S3, Amazon SQS, AWS CloudFormation, and Amazon CloudWatch. Your requirements for read-only access might differ. Several AWS services offer their own read-only managed policies, and there is also the previously mentioned AWS managed ReadOnlyAccess policy that grants read only access to all services. To customize read-only access in an identity-based policy, you can use the AWS managed

policies as a starting point and limit the actions to the services that your organization uses. The customized identity-based policy for Example Corp's `DeveloperReadOnlyRole` role is shown below.

**Policy 3: Identity-based policy attached to a developer read-only role to support human access and troubleshooting**

```
                "ec2:Get*",
                "ec2:List*",
                "ec2:Search*",
                "s3:Describe*",
                "s3:Get*",
                "s3:List*",
                "sqs:Get*",
                "sqs:List*",
                "logs:Describe*",
                "logs:FilterLogEvents",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

The CI/CD pipeline role has broad access to the account to create resources. Access to deploy through the CI/CD pipeline should be tightly controlled and monitored. The CI/CD pipeline is allowed to create new IAM roles for use with the application, but those roles are limited to only the actions allowed by the previously discussed permissions boundary. The roles, policies, and EC2 instance profiles that the pipeline creates should also be restricted to specific role paths. This enables you to enforce that the pipeline can only modify roles and policies or pass roles that it has created. This helps prevent the pipeline, and roles created by the pipeline, from elevating privileges by modifying or passing a more privileged role. Pay careful attention to the role and policy paths in the Resource element of the following CI/CD pipeline role policy (Policy 4). The CI/CD pipeline role policy also provides some example statements that allow the passing and creation of a limited set of service-linked roles (which are created in the path `/aws-service-role/`). You can add other service-linked roles to these statements as your organization adopts additional AWS services.

**Policy 4: Identity-based policy attached to CI/CD pipeline role**

```
            "iam:DeletePolicy",
            "iam:CreatePolicyVersion",
            "iam:DeletePolicyVersion",
            "iam:GetPolicy",
            "iam:TagPolicy",
            "iam:UntagPolicy",
            "iam:SetDefaultPolicyVersion"
```

In addition to the three baseline roles with identity-based policies in place that you've seen so far, there's one additional IAM that the Application Team creates using the CI/CD pipeline. This is the role that the application running on the EC2 instance w use to get and put objects from the S3 buckets in Figure 1. Explicit ownership allows the Application Team to create this iden based policy that fits their needs without having to wait and depend on the Central Cloud Team. Because the CI/CD pipeline c only create roles that have the permissions boundary policy attached, Policy 5 cannot grant more access than the permission boundary policy allows (Policy 2).

If you compare the identity-based policy attached to the EC2 instance's role (Policy 5 on left) with the permissions boundary policy described previously (Policy 2 on the right), you can see that the actions allowed by the EC2 instance's role are also allo

s3:GetObject and s3:PutObject actions. Access to create a bucket would be denied even if the role attached to the EC2 instance was given permission to perform the `s3:CreateBucket` action because the `s3:CreateBucket` action exceeds the permissions allowed by the permissions boundary.

Policy 5: Identity-based policy bound by permissions boundary and attached to the application's EC2 instance

```
{
"Id": "ApplicationRolePolicy",
"Version": "2012-10-17",
"Statement": [{
 "Effect": "Allow",
 "Action": [
    "s3:PutObject",
    "s3:GetObject"
 ],
 "Resource": "arn:aws:s3:::DOC-EXAMPLE-
 BUCKET1/*"
},
{
 "Effect": "Allow",
 "Action": [
    "s3:GetObject"
 ],
 "Resource": "arn:aws:s3:::DOC-EXAMPLE-
 BUCKET2/*"
```

Policy 2: Permissions boundary policy attached to IAM roles created by the CI/CD pipeline.

```
{
    "Id": "PermissionsBoundaryPolicy"
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "sqs:ChangeMessageVisibility",
            "sqs:DeleteMessage",
            "sqs:ReceiveMessage",
            "sqs:SendMessage",
            "sqs:PurgeQueue",
            "sqs:GetQueueUrl",
            "logs:PutLogEvents"
         ],
        "Resource": "*"
    }]
}
```

## Resource-based policies

The only resource-based policy needed in this example is attached to the bucket in the account external to the application account (DOC-EXAMPLE-BUCKET2 in the data lake account in Figure 1). Both the identity-based policy and resource-based policy must grant access to an action on the S3 bucket for access to be allowed in a cross-account scenario. The bucket policy below only allows the `GetObject` action to be performed on the bucket, regardless of what permissions the application's role (`ApplicationRole`) is granted from its identity-based policy (Policy 5).

This resource-based policy is owned by the Data Lake Team that owns and manages the data lake account (222222222222) and the policy (Policy 6). This allows the Data Lake Team to have complete control over what teams external to their AWS account can access their S3 bucket.

**Policy 6: Resource-based policy attached to S3 bucket in external data lake account (222222222222)**

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Principal": {
            "AWS": "arn:aws:iam::111111111111:role/application-roles/ApplicationRole"
        },
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"
    }]
}
```

No resource-based policy is needed on the S3 bucket in the application account (DOC-EXAMPLE-BUCKET1 in Figure 1). Access for the application is granted to the S3 bucket in the application account by the identity-based policy on its own. Access can be granted by either an identity-based policy or a resource-based policy when access is within the same AWS account.

## Putting it all together

Figure 2 shows the architecture and includes the seven different policies and the resources they are attached to. The table that follows summarizes the various IAM policies that are deployed to the Example Corp AWS environment, and specifies what team is responsible for each of the policies.

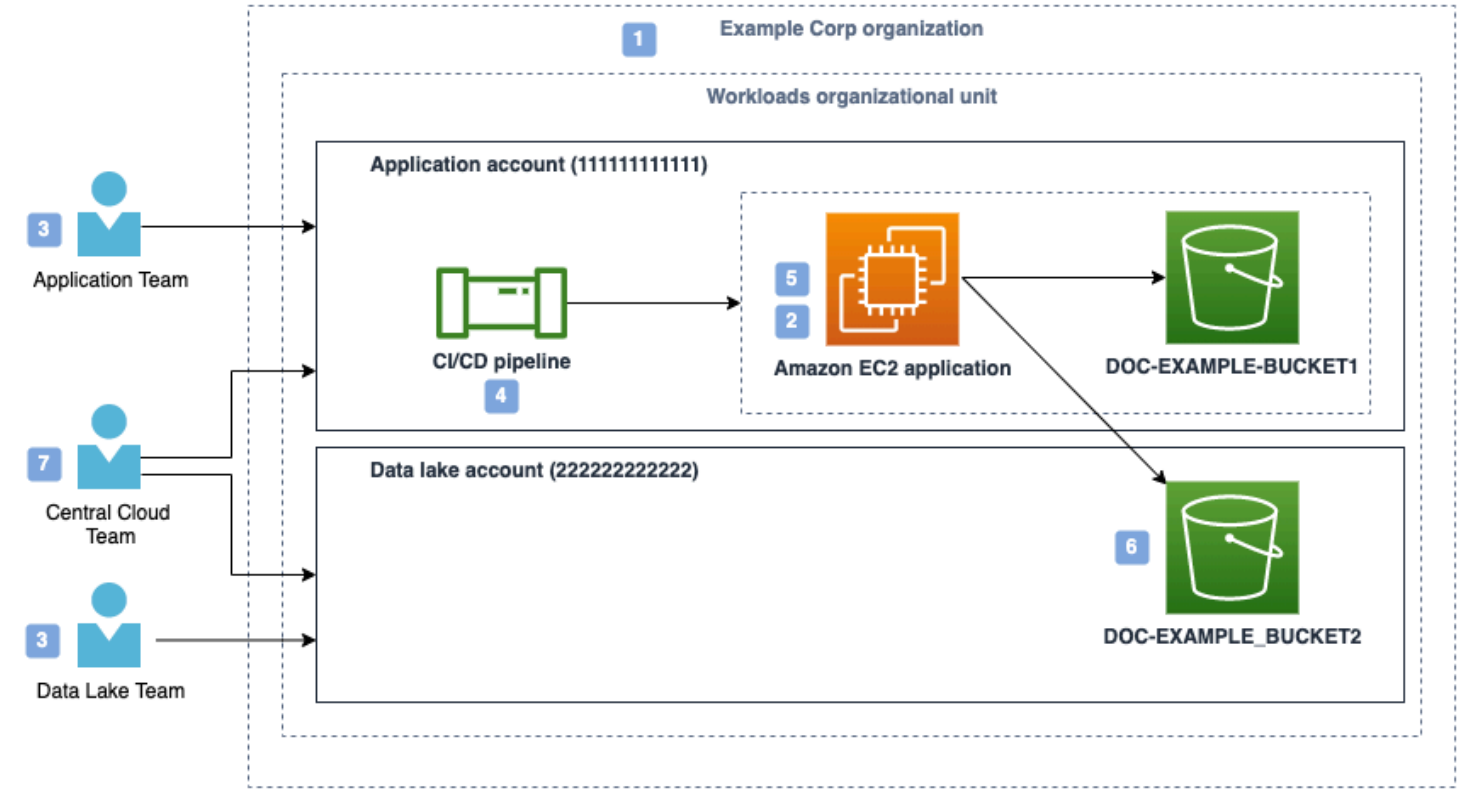


Figure 2: Sample application architecture with CI/CD pipeline used to deploy infrastructure

The numbered policies in Figure 2 correspond to the policy numbers in the following table.

| Policy number | Policy description | Policy type | Policy owner | Attached to |
|---|---|---|---|---|
| 1 | Enforce SSL and prevent member accounts from leaving the organization for all principals in the organization | Service control policy (SCP) | Central Cloud Team | Organization root |
| 2 | Restrict maximum permissions for roles created by CI/CD pipeline | Permissions boundary | Central Cloud Team | All roles created by the pipeline (`ApplicationRole`) |
| 3 | Scoped read-only policy | Identity-based policy | Central Cloud Team | `DeveloperReadOnlyRole` IAM role |
| 4 | CI/CD pipeline policy | Identity-based policy | Central Cloud Team | `CICDPipelineRole` IAM role |
| 5 | Policy used by running application to read and write to S3 buckets | Identity-based policy | Application Team | `ApplicationRole` on EC2 instance |
| 6 | Bucket policy in data lake account that grants access to a | Resource-based policy | Data Lake Team | S3 Bucket in data lake account |

| | | | | |
|---|---|---|---|---|
| | role in application account | | | |
| 7 | Broad read-only policy | Identity-based policy | Central Cloud Team | `CentralCloudTeamReadonlyRole` IAM role |

## Conclusion

In this blog post, you learned about four different policy types: identity-based policies, resource-based policies, service control policies (SCPs), and permissions boundary policies. You saw examples of situations where each policy type is commonly applied. Then, you walked through a real-life example that describes an implementation that uses these policy types.

You can use this blog post as a starting point for developing your organization's IAM strategy. You might decide that you don't need all of the policy types explained in this post, and that's OK. Not every organization needs to use every policy type. You might need to implement policies differently in a production environment than a sandbox environment. The important concepts to take away from this post are the situations where each policy type is applicable, and the importance of explicit policy ownership. We also recommend taking advantage of policy validation in AWS IAM Access Analyzer when writing IAM policies to validate your policies against IAM policy grammar and best practices.

For more information, including the policies described in this solution and the sample application, see the how-and-when-to-use-aws-iam-policy-blog-samples GitHub respository. The repository walks through an example implementation using a CI/CD pipeline with AWS CodePipeline.

If you have any questions, please post them in the AWS Identity and Access Management re:Post topic or reach out to AWS Support.

**Want more AWS Security news? Follow us on Twitter.**

TAGS: IAM, Security, Security Blog

## Comments

ALSO ON **AWS SECURITY BLOG**

**How to implement client certificate ...**

5 months ago • 2 comments

As you design your Amazon API Gateway applications to rely on mutual certificate ...

**Detecting and remediating ...**

a month ago • 1 comment

For businesses, particularly those in highly regulated industries, managing user ...

**AWS Security Profile: Tom ...**

6 months ago • 1 comment

In the AWS Security Profile series, we feature the people who work in Amazon Web ...

**Download AWS Security Hub CSV ...**

6 months ago • 9 comments

AWS Security Hub provides a comprehensive view of your security posture in ...

## 0 Comments

Start the discussion...

♡ **Share**

Be the first to comm

**Subscribe** **Privacy** **Do Not Sell My Data**