Building A Data Perimeter on AWS

AWS Whitepaper

Building A Data Perimeter on AWS: AWS Whitepaper

Table of Contents

Building a Data Perimeter on AWS

Publication date: **April 26, 2022** (*Document history* (p. 39))

Many organizations want to implement perimeter controls to help protect against unintended access and configuration errors through always-on guardrails. This paper outlines the best practices and available services for creating a perimeter around your identities, resources, and networks in AWS.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

In traditional, on-premises data center environments, a trusted network and strong authentication are the foundation of security. They establish a high-level perimeter to help prevent untrusted entities from coming in and data from going out. This perimeter provides a clear boundary of trust and ownership. When customers think about creating an AWS perimeter as part of their responsibility for security “in the cloud” in the AWS Shared Responsibility Model, they want to achieve the same outcomes. They want to draw a circle around their AWS resources, like Amazon Simple Storage Service (S3) buckets and Amazon Simple Queue Service (SQS) queues, that clearly separates “my AWS” from other customers.

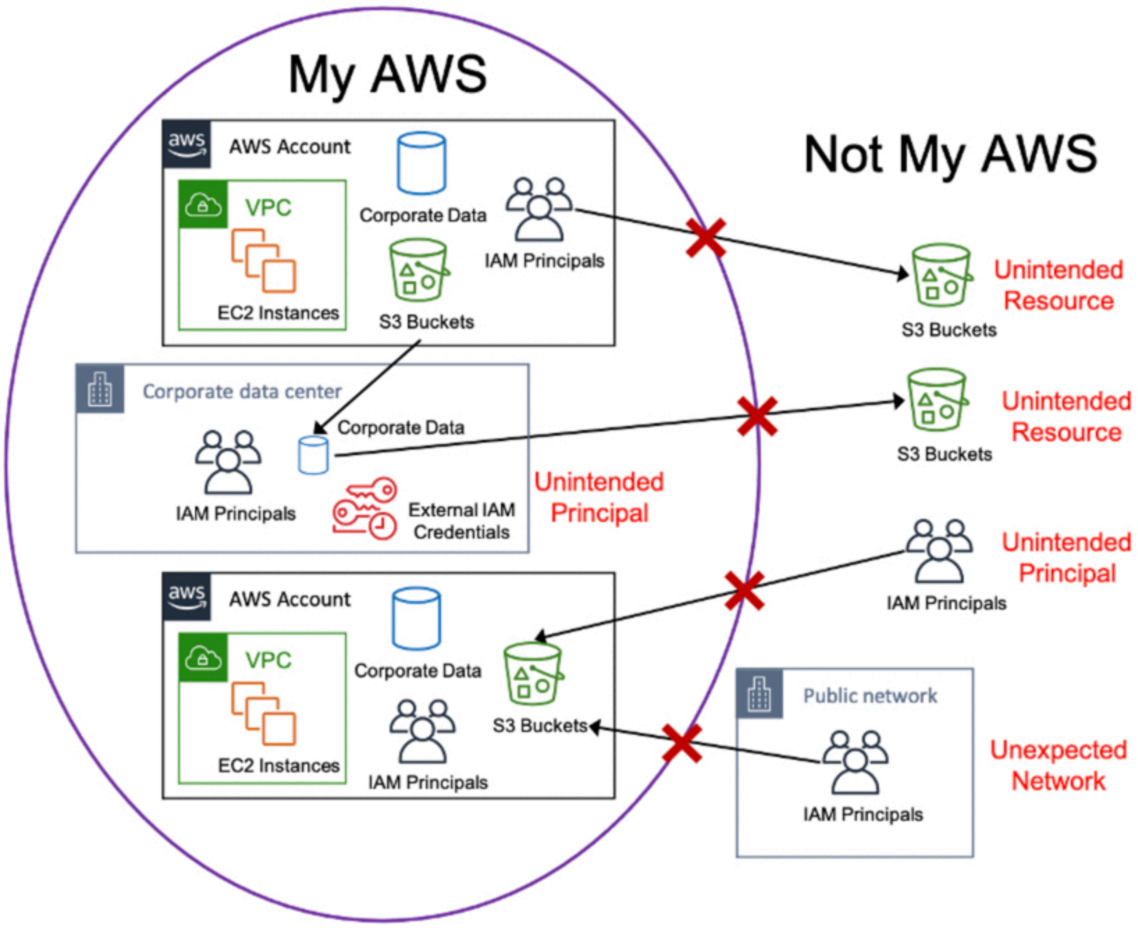
The circle that defines an AWS perimeter is typically represented as an AWS *organization* managed by AWS Organizations. AWS Organizations is an account management service that lets you consolidate multiple AWS accounts into an organization that you create and centrally manage.

Each AWS account you own is a logical container for AWS identities, resources, and networks. The AWS organization is a grouping of all of those items into a single entity. Along with on-premises networks and systems that access AWS resources, it is what most customers think of as the perimeter of “my AWS”.

The perimeter defines the things you “intend” or “expect” to happen. It refers to the access patterns among your identities, resources, and networks that should be allowed. Using those three elements, we want to make the following assertion to define our perimeter’s goal: access is allowed if the identity is trusted, the resource is trusted, and the network is expected.

If any of these conditions are false, then the access inside the perimeter is “unintended” and should be denied. The perimeter is composed of controls implemented on your identities, resources, and networks to ensure the necessary conditions are true.

This paper discusses the perimeter objectives and how the applied controls prevent unintended access patterns, particularly to data. It is designed to help customers understand how to create a complete AWS data perimeter as part of their responsibility in the AWS Shared Responsibility Model.



A high-level depiction of defining a perimeter around your AWS resources to prevent interaction with unintended AWS Identity and Access Management (IAM) principals, unintended resources, and unexpected networks

Perimeter overview

The following section provides an overview of a data perimeter's objectives and the AWS services used to implement it.

Topics

Perimeter objectives

The goal of an AWS perimeter is to ensure access is allowed if and only if an authorization involves:

- **Only trusted identities** - The AWS Identity and Access Management (IAM) principals in my AWS organization or AWS acting on my behalf. Throughout this document, references to "my principals" refers to both of these situations.
- **Only trusted resources** - The resources in my AWS organization or resources AWS operates on my behalf. Throughout this document, references to "my resources" refers to both of these situations.
- **Only expected networks** - My VPC and on-premises networks or the networks AWS uses on my behalf. Throughout this document, references to "my networks" refers to both of these situations.

These are the necessary (but not sufficient) authorization conditions for access inside an AWS perimeter to be allowed.

$$\text{Access in the Perimeter} \Rightarrow (\text{Trusted Identity}) \land (\text{Trusted Resource}) \land (\text{Expected Network})$$

Ensuring the truth of these three conditions ultimately defines the objectives of the perimeter. Each authorization condition has two objectives.

Table 1 — Authorization conditions and perimeter objectives

Authorization condition	Perimeter objective
Only trusted identities	Ensure that my resources can be accessed by only **trusted identities**.
	Ensure that only **trusted identities** are allowed from my networks.
Only trusted resources	Ensure that my identities can access only **trusted resources**.
	Ensure that only **trusted resources** can be accessed from my networks.
Only expected networks	Ensure that my identities can access resources only from **expected networks**.
	Ensure my resources can only be accessed from **expected networks**.

AWS services

You can establish a data perimeter by using permissions guardrails that restrict access outside of an organization boundary. This is achieved using three primary AWS capabilities, AWS Organizations service control policies (SCP), resource-based policies, and VPC endpoint policies. The following is a summary of these different types of policies and some considerations.

Service control policies

SCPs are a type of organization policy that you can use to manage permissions in your organization and control the maximum available permissions for your principals. SCPs offer central control over the maximum available permissions for all accounts in your organization. SCPs configured as deny lists can limit the scope of access to resources or source networks in your organization or within specific accounts. It's important to note that SCPs do not apply to service-linked roles (SLR) or AWS service principals.

Although you can define similar guard rails with IAM identity-based policies, the solutions in this document will favor using SCPs. They are a more scalable way to implement perimeter guardrails for all your IAM principals than creating and managing individual policies for every IAM principal you own. However, access to resources in AWS requires explicit permissions in identity-based policies, so they are still a necessary component of providing access, but not as part of establishing data perimeter guardrails.

Resource-based policies

These policies can be applied to AWS resources and define which IAM principals (which includes SLRs and AWS service principals) can interact with the resource, as well as what the expected networks are for access. This means that in addition to authorization through identity-based policies, these resources can define an access policy that is directly associated with the resource. These are commonly used to provide cross-account access, and can be used to authorize external AWS credentials or anonymous access. Although resource-based policies do not allow unintended access by default, a misconfigured policy might unintentionally grant access to an unintended principal or unexpected network. For a list of services that support resource-based policies, refer to AWS services that work with IAM.

VPC endpoint policies

These policies are a special type of resource-based policy that you attach to an endpoint that controls access to resources when they are accessed through that VPC endpoint. An endpoint policy does not override or replace IAM user policies or service-specific policies (such as S3 bucket policies). It is a separate policy for controlling access from the endpoint to the specified service. In VPC networks, traffic is routed to VPC endpoints automatically if you are using AWS provided DNS.

For on-premises networks, you can also route AWS traffic through VPC endpoints if they are connected to AWS via AWS Direct Connect or VPN. For services that have AWS PrivateLink interface endpoints, you can route traffic to those endpoints directly from an on-premises network. When using Amazon DynamoDB that only provides a gateway endpoint, you can use a proxy fleet as a way to route traffic from on-premises over that endpoint. For a list of services that support VPC endpoints and VPC endpoint policies, refer to AWS services that integrate with AWS PrivateLink.

Summary

The following table outlines how the different types of policies are used to achieve the perimeter objectives to support the necessary authorization conditions.

Table 2 — How different policies are used to achieve perimeter objectives

Authorization condition	**Perimeter objective**	**AWS service used**
Only trusted identities	Ensure that my resources can be accessed by only **trusted identities**.	Resource-based policies
	Ensure that only **trusted identities** are allowed from my networks.	VPC endpoint policies
Only trusted resources	Ensure that my identities can access only **trusted resources**.	SCPs
	Ensure that only **trusted resources** can be accessed from my networks.	VPC endpoint policies
Only expected networks	Ensure that my identities can access resources only from **expected networks**.	SCPs
	Ensure my resources can only be accessed from **expected networks**.	Resource-based policies

Perimeter implementation

This section describes the complete perimeter solution by evaluating each perimeter authorization condition and how the different policy types are used to achieve it. Each section describes the overall solution for that objective, provide an appendix reference with detailed policy examples, and demonstrate how the controls prevent the unintended access pattern.

Only trusted identities

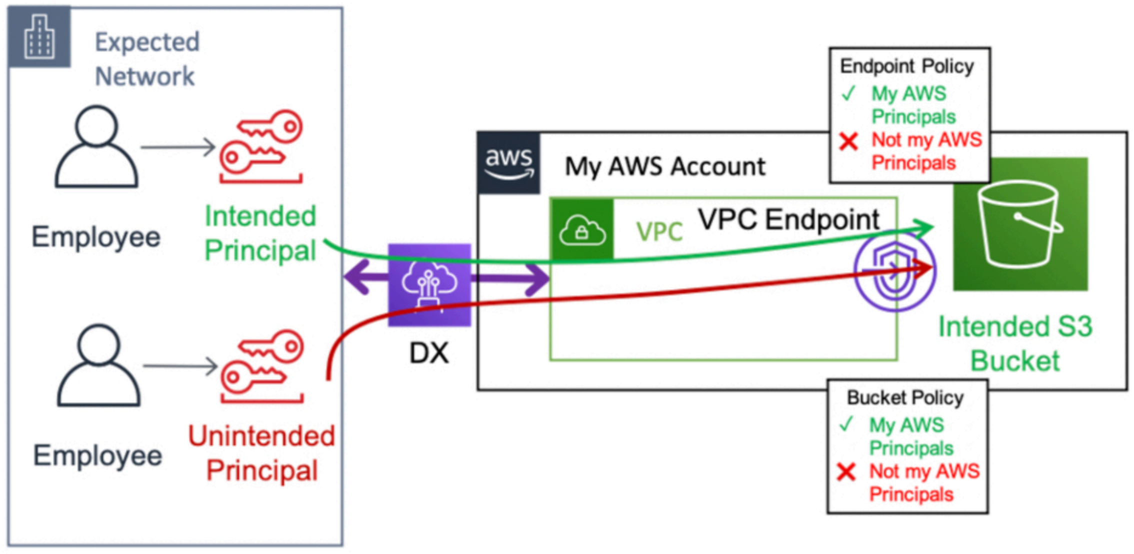
The objectives for this condition ensure that only "my principals" can access "my resources" and only "my principals" are allowed from "my networks". Resource-based policies and VPC endpoint policies will be utilized to constrain which principals are allowed access.

The primary way to ensure IAM principals belong to "my AWS" is by specifying the `aws:PrincipalOrgId` IAM policy condition in those policies. This requires that the principal being considered during the authorization of access to a resource you own or originating from a network you own (regardless of the resource owner), belongs to your AWS organization.

You can implement a more granular restriction with the `aws:PrincipalAccount` or `aws:PrincipalOrgPaths` IAM policy conditions as well. To ensure that your resource policies only allow the intended access, you can use IAM Access Analyzer for supported resources to identify resource-based policies that are too permissive.

In certain cases, AWS services may use an IAM principal that is outside of your organization, specifically a service principal, to perform actions on your behalf. For example, AWS CloudTrail uses the IAM service principal `cloudtrail.amazonaws.com` to deliver logs to your Amazon S3 bucket. These are intended actions, but need to be explicitly allowed in your resource-based policies and, in some cases, VPC endpoint policies (for example, when using a pre-signed URL for wait condition signaling from a VPC with AWS CloudFormation). You can do this using the `aws:PrincipalIsAWSService` condition.

The following diagram demonstrates how these controls help prevent unintended principals from accessing your resources or using your networks. Refer to *Appendix 1 – Resource-based policy examples* (p. 14) and *Appendix 2 – VPC endpoint policy examples* (p. 19) for a template of a standard policy statement you can add to all resource-based policies and VPC endpoint policies to achieve these perimeter objectives as well as create the necessary exceptions.

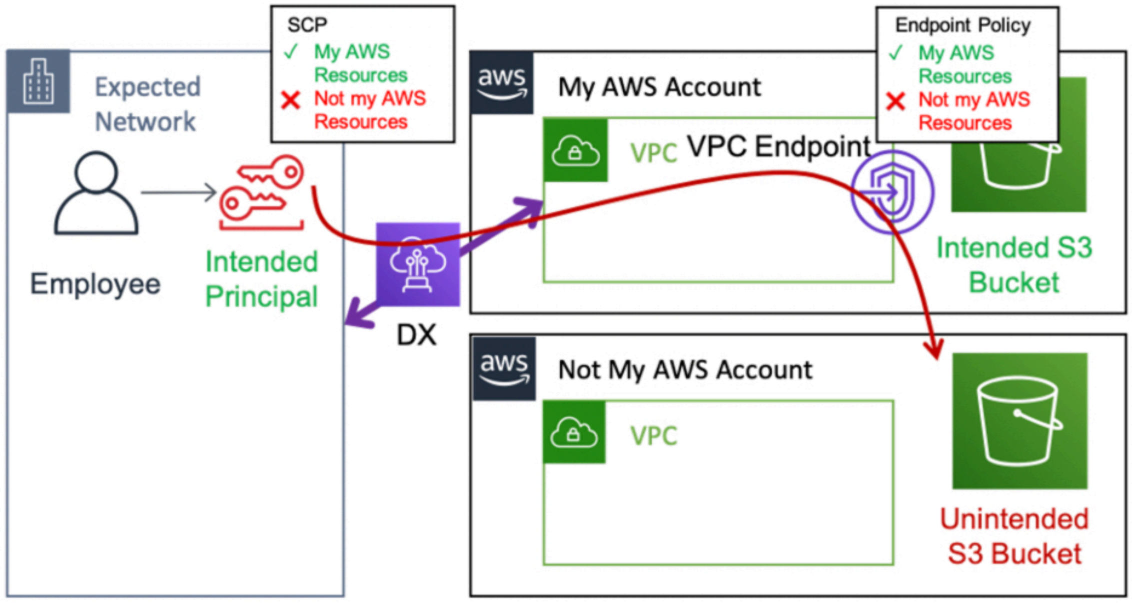


Preventing unintended principals in a resource-based policy and a VPC endpoint policy

Only trusted resources

The objectives for this condition ensure that “my principals” can only access “my resources” as well as ensures that access from “my networks” only targets “my resources” (regardless of the principal involved). SCPs and VPC endpoint policies will be utilized to constrain which resources are allowed to be accessed. Remember that SCPs don’t apply to SLRs or AWS service principals, so VPC endpoint policies will be the primary control for those entities when they operate in VPCs you own.

The primary way to ensure targeted resources belong to “my AWS” is by specifying the `aws:ResourceOrgId` IAM policy condition in SCPs and VPC endpoint policies. This ensures that the resource being considered during authorization, either being accessed directly or through a VPC endpoint, belongs to your AWS organization. The following diagram demonstrates how these policies help prevent access to an unintended resource.



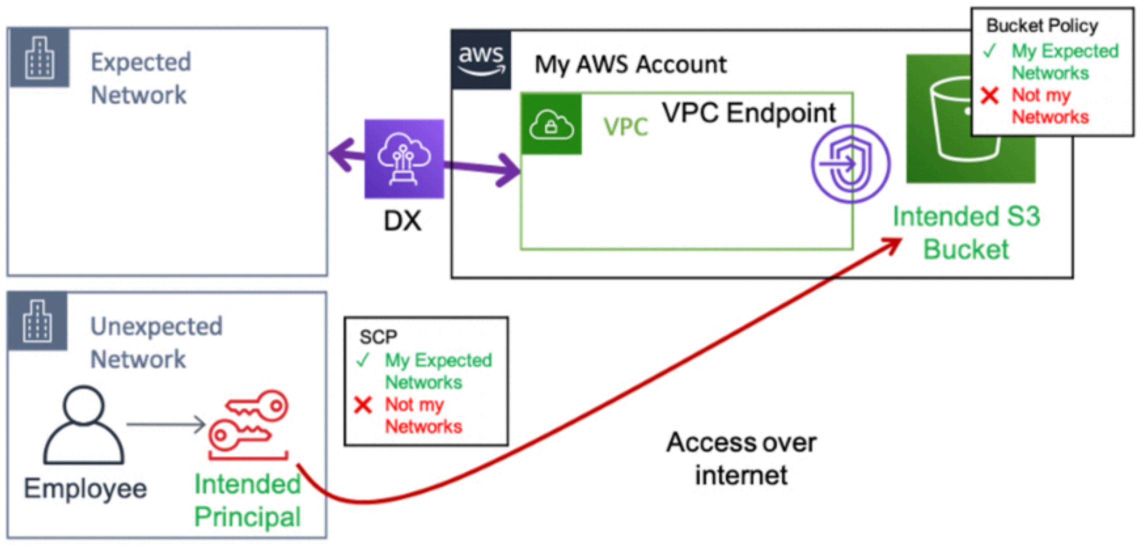
Preventing access to unintended resources with SCPs and VPC endpoint policies

Using VPC endpoint policies in this way can be considered a defense in depth approach. This is because implementing the controls for the the section called “Only trusted identities” (p. 6) objectives apply a policy on each endpoint that ensures only my principals can access resources from my networks. Then, the SCP used for this objective always applies to these principals to constrain what resources they can access. This indirectly accomplishes the same outcome as applying an `aws:ResourceOrgId` condition to your VPC endpoint policies.

Only expected networks

This final condition’s objectives ensures that only “my networks” can be the source of requests from “my principals” and/or to “my resources”. SCPs and resource-based policies will be utilized to constrain which networks are allowed for access. Remember that SCPs don’t apply to SLRs or AWS service principals, so the primary control for these entities are resource-based policies.

Within an SCP you can define the expected networks via IP address with the `aws:SourceIp` IAM policy condition or via VPC identifier with the `aws:SourceVpc` condition. Refer to *Appendix 3 – Service control policy examples* (p. 22) and *Appendix 1 – Resource-based policy examples* (p. 14) for explanations of these constraints. The following diagram shows how these policies help prevent access from unexpected network locations.



Preventing access from unexpected networks with SCPs and resource-based policies

Applying a similar constraint with a resource-based policy can also be considered a defense in depth approach. This is because the the section called "Only trusted identities" (p. 6) objectives apply a constraint on each resource-based policy that ensures only my principals can access those resources. Then, the network boundary SCP always applies to these principals to constrain the networks they can access resources from. This indirectly accomplishes the same outcome as applying an `aws:SourceIp` condition or `aws:SourceVpc` condition to your resource-based policies.

There are several scenarios where AWS will act on your behalf with your IAM credentials from networks that AWS owns that will require exceptions to these policies. For example, AWS CloudFormation provides the ability for customers to define a template of resources for which AWS orchestrates the creation, update, and deletion. The initial request to create a CloudFormation stack will originate from an expected network, but the subsequent requests for each resource in the template are made by CloudFormation in an AWS network using your original credentials (unless you've specified a service role for CloudFormation to use).

The `aws:ViaAWSService` IAM policy condition provides a way to implement an exception for some of these common scenarios where your IAM credentials are used in requests made by AWS on your behalf. *Appendix 3 – Service control policy examples* (p. 22) includes details on how to write such exceptions.

The last consideration in implementing network controls is AWS services that operate in compute environments that are not part of your network. For example, Lambda functions or SageMaker Studio Notebooks both provide an option to run in AWS-owned networks.

Some of these services provide a configuration option for running the service in your VPC as well. If you want to use the same VPC network boundary for these services, you should monitor and - where possible, enforce it - using the VPC configuration.

For example, customers can enforce AWS Lambda function deployments and updates to use Amazon Virtual Private Cloud (Amazon VPC) settings with IAM condition keys, use AWS Config Rules to audit this configuration, and then implement remediation with AWS Config Remediation Actions and AWS Systems Manager Automation documents.

It is important to note that not all AWS services are hosted as an AWS-owned endpoint authorized with IAM, for example, Amazon Relational Database Service (RDS) databases. Instead, these services expose their *data plane* inside a customer VPC.

The data plane is the part of the service that provides the day-to-day functionality of that thing. For MySQL RDS, it would be the IP address of the RDS instance on port 3306. Network controls such as firewalls or security groups should be used as part of your network boundary to prevent access to AWS services that are hosted in customer VPCs, but are not authorized with IAM credentials. Additionally, customers should leverage alternative authentication and authorization systems to access those services, such as AWS Secrets Manager for RDS access, when possible.

Mobile devices

In on-premises networks, there are some resources that are physically static, such as servers. Other resources such as laptops, however, are inherently mobile and can connect to networks outside of your control.

For example, a laptop could be connected to a corporate network when accessing data, which is temporarily stored locally, but then joins a public Wi-Fi network and sends the data to a personal Amazon S3 bucket. This network access pattern could allow access to unintended resources by bypassing corporate network controls and is a use case that you will need to consider with care.

Customers have generally tried to solve this problem with preventative controls such as always-on VPNs to keep devices connected to a corporate network. They also use detective controls (including agents) to monitor traffic and identify when preventative controls are disabled.

However, these controls aren’t fool-proof. There is still some risk that the device could join non-corporate networks. Virtual Desktop Infrastructure (VDI) is typically implemented when the risk of being able to operate a device outside of a controlled network is unacceptable and requires forcing access to AWS resources from non-mobile assets.

Amazon WorkSpaces offers a virtual desktop infrastructure (VDI) solution that can be used to require users, developers, and data scientists to use a static asset to interact with AWS resources that is subject to the same network controls as other resources in AWS VPCs. VDI solutions can also be operated by customers natively using Amazon Elastic Compute Cloud (EC2) instances in a VPC.

Additional considerations

There are a few additional considerations for specific scenarios when building a data perimeter on AWS.

Amazon S3 resource considerations

Amazon S3 is widely used to store and present publicly available website content and public data sets. Access to this content is typically performed anonymously, meaning that the HTTP requests do not have an authorization header or query string parameter generated from AWS credentials.

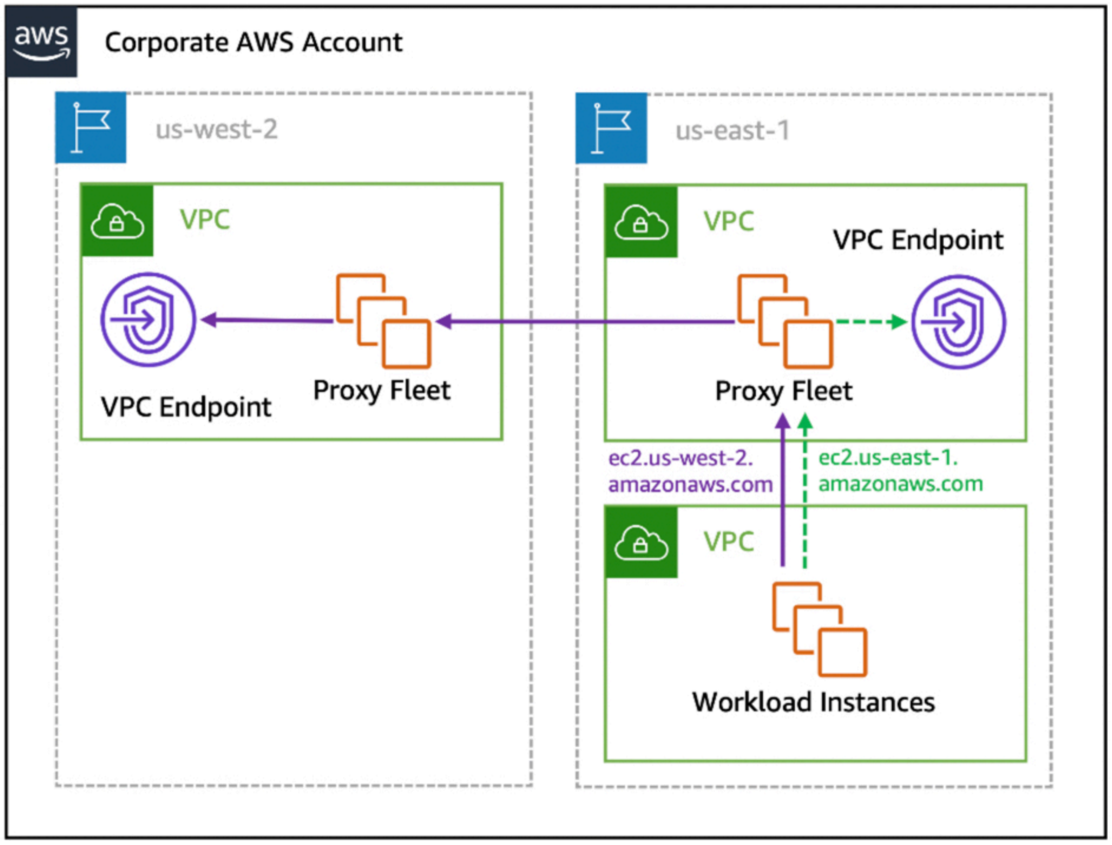
Customers may need this anonymous access for users to browse internet websites from VPC networks or on-premises networks that are routed through VPC endpoints. It is also used for workloads that may need to access public data (such as package repositories hosted on Amazon S3 or agent downloads). In order to allow this type of access, customers may choose to allow anonymous GetObject API calls in their VPC endpoint policies. This is true whether the Amazon S3 content is being accessed using the virtual or path style endpoints or is being accessed via an Amazon S3 website endpoint.

Access to all other Amazon S3 APIs should be authenticated. *Appendix 2 – VPC endpoint policy examples* (p. 19) also includes details of how to allow anonymous `GetObject` API calls while enforcing authentication and guardrails for intended resources for the remainder of Amazon S3 actions.

Cross-Region requests

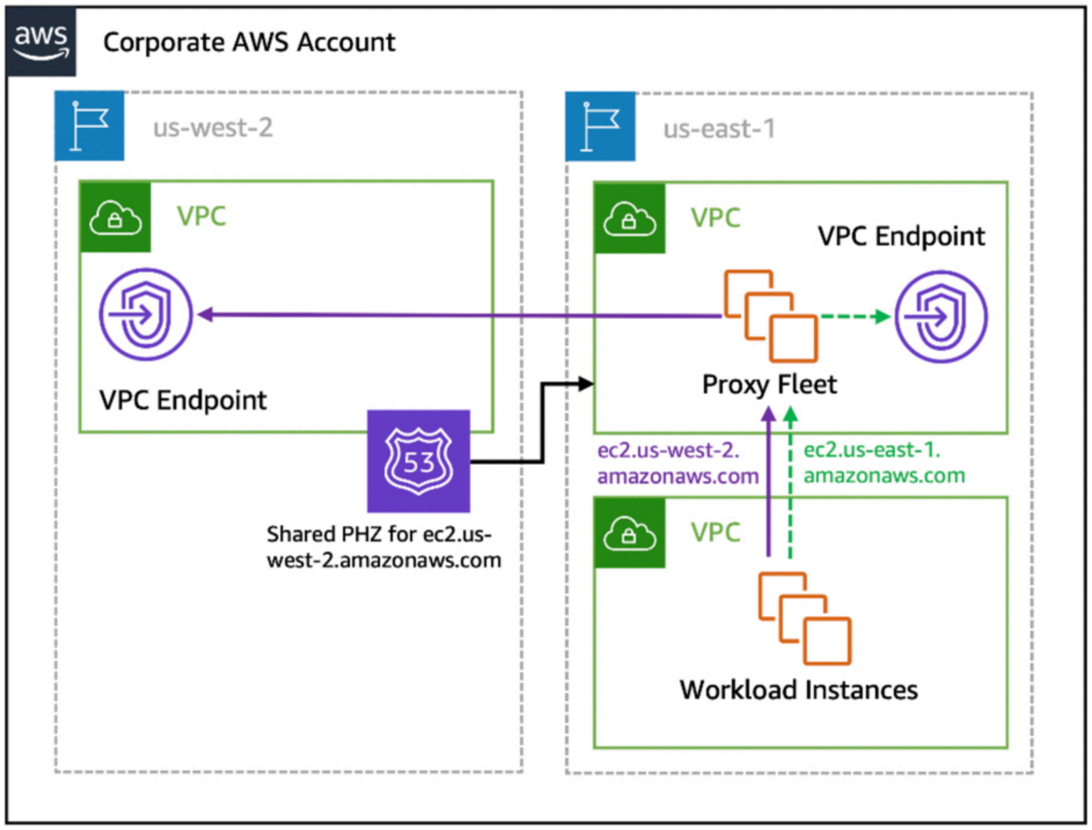
VPC endpoints only support routing AWS API calls to service endpoints that are in the same Region as the VPC endpoint itself. For example, an Amazon S3 VPC endpoint in a VPC in us-east-1 only supports routing traffic for requests made to S3 buckets in us-east-1. A call to PutObject for a bucket in us-west-2 would not traverse the VPC endpoint and would not have the endpoint policy applied to the request. To ensure the intended security controls are applied consistently, you can handle cross-Region requests in three ways.

- **Prevent cross-Region API calls using a proxy**. This does not require inspecting TLS and can be done by looking at the hostname in the `CONNECT` request or, if using Server Name Indication (SNI), the hostname presented in the `ClientHello`, since the AWS Region is included in the domain name of the URL (with the exception of some services that only provide a single control plane endpoint such as IAM or Route 53).
- **Use a centralized VPC endpoint approach for each Region and share Route 53 private hosted zones (PHZ) for the endpoints**. This allows instances to resolve the out-of-Region endpoint domain name locally and send their request directly to the VPC endpoint. This pattern is described in more detail in Centralized access to VPC private endpoints.
- If you use HTTP/S proxies in your environment, you can use them to **forward out-of-Region requests**. There are two variations for this option:
  - The first variation uses *proxy-chaining*. The proxy in the local Region forwards traffic to a peer proxy running in a VPC in the destination Region. The out-of-Region proxy delivers the traffic to the appropriate VPC endpoint in its Region. Refer to *Appendix 4 – Proxy configuration examples* (p. 27) for an example proxy configuration that implements this *proxy-chaining* solution. The following diagram demonstrates a high-level reference architecture.



Using proxy-chaining to send out-of-Region requests through VPC endpoints

- Implement a centralized VPC endpoint approach using shared Route 53 private hosted zones as described in option 2. The local proxy uses AWS-provided VPC DNS and sends the request directly to the out-of-Region VPC endpoint. This eliminates the need to configure proxy-chaining, but does require the creation of a private hosted zone (PHZ) for each endpoint that will need to be shared with every VPC hosting a proxy.



Forwarding out-of-Region requests using a shared Route 53 PHZ

Preventing access to temporary credentials

Except for the cases of credential theft or leakage, the only other way for an unintended entity to gain access to temporary credentials derived from IAM roles that are part of “my AWS” is through misconfigured IAM role trust policies.

IAM role trust policies define the principals that you trust to assume an IAM role. A role trust policy is a required resource-based policy that is attached to a role in IAM. The principals that you can specify in the trust policy include users, roles, accounts, and services.

The trust policy can be configured to help ensure that no one from outside the customer’s account or organization can be authorized to assume the role. Customers should audit all IAM role trust policies and ensure either of the following are true:

- The trust policy uses either the `aws:PrincipalOrgId` or `aws:PrincipaOrgPaths` condition when the trusted entity is an IAM principal, such as a role or user. Exceptions can be created with an allow list of known, external, expected accounts and they should use the `sts:ExternalId` condition.

- The trusted entity is an AWS service, being either a service principal or IAM service-linked role. As a best practice, the trust policy should not trust more than one AWS service in order to apply least privilege.

Refer to *Appendix 5 – IAM role trust policy example* (p. 30) for more details.

Resource sharing and external targets

The final consideration are services that allow resource sharing or targeting external resources. With these services, you cannot use a condition such as `aws:ResourceOrgId` because the resource being evaluated in the policy belongs to your AWS organization, but its configuration specifies a resource that does not. Instead, you will need to use different approaches to prevent sharing resources with AWS accounts outside of your organization. The following is a list of several AWS services that allow you to share a resource, such as an EBS snapshot, with another account or target a resource in another account.

- **Amazon Machine Images (AMI)** - AMIs can be shared with other accounts or made public with the `ModifyImageAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **Amazon EBS Snapshots** - EBS Snapshots can be shared with other accounts or made public with the `ModifySnapshotAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **Amazon RDS Snapshots** - RDS Snapshots can be shared with other accounts or made public with the `ModifyDBSnapshotAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **AWS Resource Access Manager (RAM)** - RAM is a service that allows sharing various types of resources with other AWS accounts. Sharing with RAM can be constrained to your AWS organization using the `ram:RequestedAllowsExternalPrincipals` IAM condition.
- **Amazon CloudWatch Logs Subscription Filters** - You can send CloudWatch Logs to cross account destinations. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **Amazon EventBridge Targets** - You can add targets to an Amazon EventBridge rule that are in different accounts than the event bus. Use the `events:TargetArn` IAM condition to limit which accounts can be used as targets for EventBridge rules.

AWS RAM allows for conditioning on sharing outside of your AWS organization in an SCP. For the rest of the services, you can use IAM policies to prevent the actions altogether. However, this isn’t always practical. In these cases, you can either allow just privileged roles to take those actions, or you can build detective controls and remediate the configuration using services such as AWS Config or EventBridge.

For example, an EventBridge rule can look for `PutSubscriptionFilter` events and invoke a Lambda function to evaluate the destination ARN in the subscription. If the ARN is for a resource not in the AWS organization, the function can remove the subscription. Refer to *Appendix 6 - Resource sharing and external targets policy examples* (p. 32) for more detailed examples of policies you can use to prevent resource sharing.

Conclusion

This paper has reviewed how to implement a data perimeter on AWS using SCPs, resource-based policies, and VPC endpoint policies. These controls are used to ensure **Only Trusted Identities**, **Only Trusted Resources**, and **Only Expected Networks** are allowed access to "my AWS".

The following is a list of the recommendations made throughout this paper as part of achieving the perimeter's six objectives.

- Use the `aws:PrincipalOrgId` condition in resource-based policies and VPC endpoint policies to help prevent unintended principals.
- Use the `aws:ResourceOrgId` condition in SCPs to help prevent your IAM principals from accessing unintended resources. Additionally, add this condition to your VPC endpoint policies as a defense in depth approach.
- Use an SCP to prevent access from unexpected network locations. Additionally, add similar policy statements to your resource-based policies as a defense in depth approach. Use the `aws:ViaAWSService` condition to create exceptions when AWS acts on your behalf using your credentials.
- Audit all resource-based policies and VPC endpoint policies to ensure that guardrail controls are applied to help prevent misconfiguration. Use IAM Access Analyzer to review resource-based policy configuration. Use the `aws:PrincipalIsAWSService` condition to create exceptions in resource-based policies and VPC endpoint policies for AWS services.
- Block all outbound Internet access, except for required AWS endpoints and allowed external services that are dependencies for your workloads. This prevents data movement to non-AWS destinations, out-of-Region AWS endpoints, and unintended VPC hosted data plane services (like RDS instances).
- Route out-of-Region requests through VPC endpoints so that the network boundary controls are consistently applied.
- Where AWS provides an option to run a resource publicly or inside a customer-owned VPC, use the VPC configuration (that is, Amazon OpenSearch Service (OpenSearch Service), Amazon SageMaker notebooks, and AWS Lambda) and turn off the public access options (for example, Amazon Redshift and RDS) in order to use network controls.
- Configure IAM Role Trust Policies with condition statements, limiting access to only intended principals when the trusted entity is an IAM principal (as opposed to an AWS service principal).
- Prevent external resource sharing and targeting external resources with an SCP.

Appendix 1 – Resource-based policy examples

Only trusted identities

The following provides an example that can be used in a resource-based policy to help prevent access by unintended principals.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

There are expected situations when AWS uses an IAM service principal instead of an IAM role to interact with your resources. An example would be AWS CloudTrail log delivery to Amazon S3. The service principal is not part of your AWS organization like IAM roles are, so it needs to be excluded from the restriction. You cannot use a `NotPrincipal` statement with an AWS service principal, so instead you can use the `aws:PrincipalIsAWSService` condition. This provides an example of an S3 bucket policy for CloudTrail that ensures no one outside of your organization can access the bucket, except for the CloudTrail service principal.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipalsButAllowCloudTrail",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "false"
        }
      }
    },
    {
      "Sid": "AllowCloudTrailToGetACL",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname"
      ]
    },
    {
      "Sid": "AllowCloudTrailToPutLogs",
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname/AWSLogs/123456789012/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    }
  ]
}
```

Only expected networks

The following policy helps prevent access from unexpected networks in an S3 bucket policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnexpectedNetworksButAllowCloudTrailAndVia",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "NotIpAddressIfExists": {
          "aws:SourceIp": "203.0.113.0/24"
        },
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-012abc01"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "false",
          "aws:ViaAWSService": "false"
        },
        "Null": {
          "aws:PrincipalTag/IpRestrictedExempt": true
        }
      }
    }
  ]
}
```

The policy blocks unexpected networks using similar logic discussed more deeply in *Appendix 3 – Service control policy examples* (p. 22). For example, this exclusion using the `aws:ViaAWSService` condition would allow Amazon Athena to run queries on the CloudTrail logs. It also includes the `aws:PrincipalIsAWSService` condition in case you also wanted to use this policy statement in a situation like the above CloudTrail example.

Only trusted identities and expected networks

The two previous policies could be generalized into a complete guardrail for other buckets by removing the explicit allow statements for the CloudTrail service principal.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnexpectedNetworksButAllowVia",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "NotIpAddressIfExists": {
          "aws:SourceIp": "203.0.113.0/24"
        },
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-012abc01"
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        },
        "Null": {
          "aws:PrincipalTag/IpRestrictedExempt": true
        }
      }
    },
    {
      "Sid": "PreventUnintendedPrincipals",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

The complete example for allowing CloudTrail using the above policy template looks like the following.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnexpectedNetworksButAllowCloudTrailAndVia",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "NotIpAddressIfExists": {
          "aws:SourceIp": "203.0.113.0/24"
        },
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-012abc01"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "false",
          "aws:ViaAWSService": "false"
        },
        "Null": {
          "aws:PrincipalTag/IpRestrictedExempt": true
        }
      }
    },
    {
      "Sid": "PreventUnintendedPrincipalsButAllowCloudTrail",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::bucketname",
        "arn:aws:s3:::bucketname/*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "false"
        }
      }
    },
    {
      "Sid": "AllowCloudTrailToGetACL",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname"
      ]
    },
    {
      "Sid": "AllowCloudTrailToPutLogs",
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname/AWSLogs/123456789012/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    }
  ]
}
```

Appendix 2 – VPC endpoint policy examples

Only trusted identities

The following is an example of a VPC endpoint policy for Amazon DynamoDB that restricts access to only credentials that are part of the customer's AWS organization as a way to help prevent unintended principals.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Principal": "*",
      "Action": [
        "dynamodb:*"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

The policy could also be written with two sections in the statement, making the condition part of an explicit deny. Refer to IAM Policy Evaluation Logic for an explanation of how these policies are evaluated.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Sid": "AllowAllDynamoDB",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Only trusted resources

The following policy helps prevent access to unintended Amazon S3 resources through a VPC endpoint. You may also need to exempt certain AWS-owned Amazon S3 buckets from this policy and explicitly allow access to them depending on your use cases.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedResources",
      "Principal": "*",
      "Action": [
        "s3:*"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

It can also be written in a similar way to the previous example using two statements.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedResources",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Sid": "AllowAllS3",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Only trusted identities and resources

Finally, you can create a combined policy for a VPC endpoint that helps prevent access to unintended principals and resources.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedResourcesAndPrincipals",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "o-4tkekae453",
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Sid": "AllowAllS3",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

For a VPC endpoint policy, it's important to explicitly specify the AWS organization ID in the condition. **DO NOT** use a condition like this:

```
"Condition": {
  "StringNotEquals": {
    "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
  }
}
```

If an unintended principal in a different AWS Organization tries to access a resource in their own organization over the VPC endpoint, this condition will not deny the action.

Other services might also require exceptions to this kind of policy. For example, when you use `cfn-hup`, it supports the `on.command` hook that allows you to use Amazon SQS messages to invoke the `cfn-hup` actions. This is used by AWS Elastic Beanstalk environments where the `cfn-hup` daemon retrieves a CloudFormation specific credential to query an SQS queue owned by AWS. An SQS VPC endpoint policy needs to allow this if your EC2 instances use `cfn-hup` in this way.

Appendix 3 – Service control policy examples

Only expected networks

This policy can be applied once at the organization root level, in which case, you'll need to scale the `aws:SourceVpc` condition to include VPCs from all of your accounts. You can also apply this policy in a more granular way at an organizational unit or individual account level (meaning you would have multiple SCPs of this type deployed). Be sure to check that your SCPs conform to the quotas provided by AWS Organizations. Additionally, this policy itself is just an example of how to build a network boundary SCP, you may need to exclude additional actions based on your environment and use case. Be sure to test any SCP thoroughly before deploying to production to understand its impacts and additional exclusions that may be required.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "es:ESHttp*",
        "dax:PutItem",
        "dax:Query",
        "dax:Scan",
        "dax:GetItem",
        "dax:DeleteItem",
        "dax:UpdateItem",
        "dax:BatchGetItem",
        "dax:BatchWriteItem",
        "dax:ConditionCheckItem"
      ],
      "Resource": "*",
      "Condition": {
        "NotIpAddressIfExists": {
          "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24"
          ]
        },
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": [
            "vpc-012abc01",
            "vpc-023edf34"
          ]
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        },
        "Null": {
          "aws:PrincipalTag/IpRestrictedExempt": true
        },
        "ArnNotLike": {
          "aws:PrincipalArn": "arn:aws:iam::*:role/aws:ec2-infrastructure"
        }
      }
    }
  ]
}
```

This is a summary of the SCP contents. The policy denies all actions to all resources, except for the actions listed in the NotAction section. They are listed because Amazon DynamoDB Accelerator (DAX) and Amazon Elasticsearch Service (Amazon ES), when configured as a VPC domain, do not present a public IP address or transit a VPC endpoint and cannot be controlled with a source IP condition.

Because each condition operator is evaluated with a logical AND , every condition must evaluate to true for the policy to deny the action. Thus, any one of the conditions evaluating to false will permit the action. In that light, the conditions can be viewed as exceptions to the policy.

- `aws:SourceIp` – The action is allowed if it originates from one of the listed subnets. Customers should replace these IPs with the IPs of their NAT gateways, EIPs, and on-premises public IP space.
- `aws:SourceVpc` – When customers have VPC endpoints implemented, they should replace these values with the VPC IDs of their own VPCs.
- `aws:ViaAWSService` – Some services, such as CloudFormation, perform actions on a user's behalf by using their credentials and will not present customer IP addresses that they have listed in the "`aws:SourceIp`" condition block. This condition allows those services to still initiate those actions without being restricted to the customer network.
- `aws:PrincipalTag` – Using a standard tag on IAM principals allows customers to exempt them from this policy if needed. For example, the IAM role used by Amazon S3 Same Region or Cross Region Replication operates from an AWS-owned network and would need to be excluded from this restriction. Customers should replace the tag key with the tag that they will use to mark principals as exempt.
- `aws:PrincipalArn` – Amazon EC2 uses special infrastructure IAM roles to perform actions on customers' behalf per EC2 instance that does not present a predictable public IP address and can be safely exempted from this policy. The assumed role will look something like this: `arn:aws:sts::123456789012:assumed-role/aws:ec2-infrastructure/i-07d8bc39180cd7268`. The role name uses a ":" character, which is unallowed for normal roles, so it cannot be spoofed by another customer.

Only trusted resources

The following policy helps prevent access to unintended resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
        }
      }
    }
  ]
}
```

This policy makes use of a policy variable to match the resource's organization ID to the principal's organization ID. Using this statement in an SCP is acceptable, unlike in *Appendix 2 – VPC endpoint policy* *examples* (p. 19), because the SCP only ever applies to IAM principals that are part of the customer's AWS organization. Do not use this policy as written in a VPC endpoint policy, instead specify the actual AWS organization ID in place of the policy variable.

This policy may need exceptions when your IAM principals access resources owned by AWS (or intended resources outside of your organization). The first way this can happen is when AWS interacts with resources on your behalf. For example, if you use CloudFormation Stack Sets in your environment, Stack Sets send requests to Amazon SNS topics owned by CloudFormation on behalf of the IAM principal invoking the Stack Sets APIs. The other way this can happen is when you interact with AWS resources directly, for example, querying public SSM parameters. A policy that allows exceptions for SSM and Stack Sets follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "sns:*",
        "ssm:GetParameter*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "sns:*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
        },
        "ForAllValues:StringNotEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

Other services, such as AWS Data Exchange, that make requests to Amazon S3 on a customer's behalf, operate in a similar way to the CloudFormation example above. The policy can be generalized to trust all AWS services when they interact with a resource not inside the customer's organization using the `ViaAWSService` condition.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "ssm:GetParameter*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

Maintain the list of exempted actions that you want to allow your IAM principals to use directly with external resources as part of the `NotAction` list in the policy statement.

Only expected networks and trusted resources

You can combine the two previous policies into a single policy to achieve both objectives in an SCP.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "es:ESHttp*",
        "dax:PutItem",
        "dax:Query",
        "dax:Scan",
        "dax:GetItem",
        "dax:DeleteItem",
        "dax:UpdateItem",
        "dax:BatchGetItem",
        "dax:BatchWriteItem",
        "dax:ConditionCheckItem"
      ],
      "Resource": "*",
      "Condition": {
        "NotIpAddressIfExists": {
          "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24"
          ]
        },
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": [
            "vpc-012abc01",
            "vpc-023edf34"
          ]
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        },
        "Null": {
          "aws:PrincipalTag/IpRestrictedExempt": true
        },
        "ArnNotLike": {
          "aws:PrincipalArn": "arn:aws:iam::*:role/aws:ec2-infrastructure"
        }
      }
    },
    {
      "Effect": "Deny",
      "NotAction": [
        "ssm:GetParameter*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgId": "${aws:PrincipalOrgId}"
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

Appendix 4 – Proxy configuration example

The following configuration is for a Squid-based proxy running in us-east-1 with peers in us-west-2 and eu-west-1. It denies all other traffic for the amazonaws.com domain, but allows all other domains to be forwarded normally. This configuration will need to be kept up to date with global AWS services that do not use a Region name in their domain name.

```
cache_effective_user squid
prefer_direct off
nonhierarchical_direct off

## Define acls for local networks that are forwarding here
acl rfc_1918 src 10.0.0.0/8 # RFC1918 possible internal network
acl rfc_1918 src 172.16.0.0/12 # RFC1918 possible internal network
acl rfc_1918 src 192.168.0.0/16 # RFC1918 possible internal network
acl localnet src fc00::/7 # RFC 4193 local private network range
acl localnet src fe80::/10 # RFC 4291 link-local (directly plugged) machines
acl localnet src 127.0.0.1 # localhost loopback

## Additional ACLs
acl ssl_ports port 443 # ssl
acl safe_ports port 80 # http
acl safe_ports port 21 # ftp
acl safe_ports port 443 # https
acl safe_ports port 70 # gopher
acl safe_ports port 210 # wais
acl safe_ports port 1025-65535 # unregistered ports
acl safe_ports port 280 # http-mgmt
acl safe_ports port 488 # gss-http
acl safe_ports port 591 # filemaker
acl safe_ports port 777 # multiling http
acl CONNECT method CONNECT

## Define acls for amazonaws.com
acl aws_domain dstdomain .amazonaws.com
acl us_east_1 dstdomain .s3.amazonaws.com
acl us_east_1 dstdomain .sts.amazonaws.com
acl us_east_1 dstdomain .cloudfront.amazonaws.com
acl us_west_2 dstdomain .globalaccelerator.amazonaws.com
acl us_east_1 dstdomain .iam.amazonaws.com
acl us_east_1 dstdomain .route53.amazonaws.com
acl us_east_1 dstdomain .queue.amazonaws.com
acl us_east_1 dstdomain .sdb.amazonaws.com
acl us_east_1 dstdomain .waf.amazonaws.com
acl us_east_1 dstdomain .us-east-1.amazonaws.com
acl us_east_2 dstdomain .us-east-2.amazonaws.com
acl us_west_2 dstdomain .us-west-2.amazonaws.com
acl eu_west_1 dstdomain .eu-west-1.amazonaws.com
acl us_east_1_alt dstdom_regex \.us-east-1\..*?\.amazonaws.com
acl us_east_2_alt dstdom_regex \.us-east-2\..*?\.amazonaws.com
acl us_west_2_alt dstdom_regex \.us-west-2\..*?\.amazonaws.com
acl eu_west_1_alt dstdom_regex \.eu-west-1\..*?\.amazonaws.com

## Deny access to anything other than SSL
http_access deny !safe_ports
http_access deny CONNECT !ssl_ports
```

```
## Now specify the cache peer for each Region
never_direct allow us_east_2
never_direct allow us_east_2_alt
never_direct allow us_west_2
never_direct allow us_west_2_alt
never_direct allow eu_west_1
never_direct allow eu_west_1_alt
cache_peer us-east-2.proxy.local parent 3128 0 no-query proxy-only name=cmh
cache_peer_access cmh allow us_east_2
cache_peer_access cmh allow us_east_2_alt
cache_peer us-west-2.proxy.local parent 3128 0 no-query proxy-only name=pdx
cache_peer_access pdx allow us_west_2
cache_peer_access pdx allow us_west_2_alt
cache_peer eu-west-1.proxy.local parent 3128 0 no-query proxy-only name=dub
cache_peer_access dub allow eu_west_1
cache_peer_access dub allow eu_west_1_alt

# Only allow cachemgr access from localhost
http_access allow localhost manager
http_access deny manager

## Explicitly allow approved AWS Regions so we can block
## all other Regions using .amazonaws.com below
http_access allow rfc_1918 us_east_1
http_access allow rfc_1918 us_east_2
http_access allow rfc_1918 us_west_2
http_access allow rfc_1918 eu_west_1
http_access allow rfc_1918 us_east_1_alt
http_access allow rfc_1918 us_east_2_alt
http_access allow rfc_1918 us_west_2_alt
http_access allow rfc_1918 eu_west_1_alt

## Block all other AWS Regions
http_access deny aws_domain

## Allow all other access from local networks
http_access allow rfc_1918
http_access allow localnet

## Finally deny all other access to the proxy
http_access deny all

## Listen on 3128
http_port 3128

## Logging
access_log stdio:/var/log/squid/access.log
strip_query_terms off
logfile_rotate 1

## Turn off caching
cache deny all

## Enable the X-Forwarded-For header
forwarded_for on

## Suppress sending squid version information
httpd_suppress_version_string on

## How long to wait when shutting down squid
shutdown_lifetime 30 seconds

## Hostname
visible_hostname aws_proxy

## Prefer ipv4 over v6
```

```
dns_v4_first on
```

Appendix 5 – IAM role trust policy example

The following is an example of a policy statement you can add to an existing role trust policy document to ensure all principals that are allowed to assume the role are part of your specified organization.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sts:AssumeRole",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgId": "o-4tkekae453"
    }
  }
}
```

If you need to make an exception for a customer in account *123456789012* using an external ID of "12345", you can add the `PrincipalOrgId` condition to the statement where you allow the IAM principals in your org.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::094697565664:role/role1",
          "arn:aws:iam::094697565664:role/role2",
          "arn:aws:iam::094697565664:role/role3",
          "arn:aws:iam::094697565664:role/role4",
          "arn:aws:iam::094697565664:role/role5",
          "arn:aws:iam::094697565664:role/role6",
          "arn:aws:iam::094697565664:role/role7",
          "arn:aws:iam::094697565646:role/role8",
          "arn:aws:iam::087695765465:role/role9",
          "arn:aws:iam::087695765465:role/role10"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "12345"
        }
      }
    }
  ]
}
```

In this case, the account number for `role8` was mistyped and belongs to an account outside of the specified organization. The `PrincipalOrgId` condition will prevent `role8` from being able to assume this role while still allowing the external user.

Appendix 6 - Resource sharing and external targets policy examples

Resource Access Manager (RAM)

The following policy can be used as an SCP to prevent sharing resources outside of your AWS organization. Refer to Example service control policies for AWS Organizations and AWS RAM for additional SCP examples with RAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ram:CreateResourceShare",
        "ram:UpdateResourceShare"
      ],
      "Resource": "*",
      "Condition": {
        "Bool": {
          "ram:RequestedAllowsExternalPrincipals": "true"
        }
      }
    }
  ]
}
```

EBS snapshots

You can prevent sharing EBS snapshots in an SCP and create an exception for a privileged IAM principal, if required in your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ec2:ModifySnapshotAttribute"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    }
  ]
}
```

For IAM roles that you want to be exempt from this deny statement, add a tag key of *DenySharingExempt* (update the key name to fit your needs) with any tag value.

RDS snapshots

You can prevent sharing RDS snapshots in an SCP and create an exception for a privileged IAM principal, if required in your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "rds:ModifyDBSnapshotAttribute"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    }
  ]
}
```

For IAM roles that you want to be exempt from this deny statement, add a tag key of *DenySharingExempt* (update the key name to fit your needs) with any tag value.

AMIs

You can prevent sharing AMIs in an SCP and create an exception for a privileged IAM principal, if required in your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ec2:ModifyImageAttribute"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    }
  ]
}
```

For IAM roles that you want to be exempt from this deny statement, add a tag key of *DenySharingExempt* (update the key name to fit your needs) with any tag value.

Subscription filters

You can prevent creating CloudWatch subscription filters in an SCP and create an exception for a privileged IAM principal, if required in your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "logs:PutSubscriptionFilter"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    }
  ]
}
```

For IAM roles that you want to be exempt from this deny statement, add a tag key of *`DenySharingExempt`* (update the key name to fit your needs) with any tag value.

Amazon EventBridge targets

You can prevent creating EventBridge targets in an SCP and create an exception for a privileged IAM principal, if required in your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "events:PutTargets"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    }
  ]
}
```

For IAM roles that you want to be exempt from this deny statement, add a tag key of *`DenySharingExempt`* (update the key name to fit your needs) with any tag value. Alternatively, you can specify Amazon Resource Names (ARNs) with a wildcard to allow list specific accounts.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "events:PutTargets"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "events:TargetArn": [
            "arn:aws:*:*:123456789012:*",
            "arn:aws:*:*:111122223333:*"
          ]
        }
      }
    }
  ]
}
```

Combined policy

The following is a combined policy to help prevent resource sharing.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
          "ram:CreateResourceShare",
          "ram:UpdateResourceShare"
      ],
      "Resource": "*",
      "Condition": {
        "Bool": {
          "ram:RequestedAllowsExternalPrincipals": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "ec2:ModifySnapshotAttribute",
        "ec2:ModifyImageAttribute",
        "rds:ModifyDBSnapshotAttribute",
        "logs:PutSubscriptionFilter"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:PrincipalTag/DenySharingExempt": true
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "events:PutTargets"
      ],
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "events:TargetArn": [
            "arn:aws:*:*:123456789012:*",
            "arn:aws:*:*:111122223333:*"
          ]
        }
      }
    }
  ]
}
```

Contributors

Contributors to this document include:

- Michael Haken, Principal Solutions Architect, Amazon Web Services

Further reading

For additional information, refer to:

- IAM makes it easier for you to manage permissions for AWS services accessing your resources (blog)
- Data Perimeter Workshop
- Centralized access to VPC private endpoints

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated (p. 39)	Updated to add guidance for using `aws:ResourceOrgId`, added additional policy examples, and organizational updates.	April 26, 2022
Whitepaper updated (p. 39)	Content and policy example updates.	September 8, 2021
Initial publication (p. 39)	Whitepaper first published.	July 1, 2021

Note
To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

AWS glossary

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