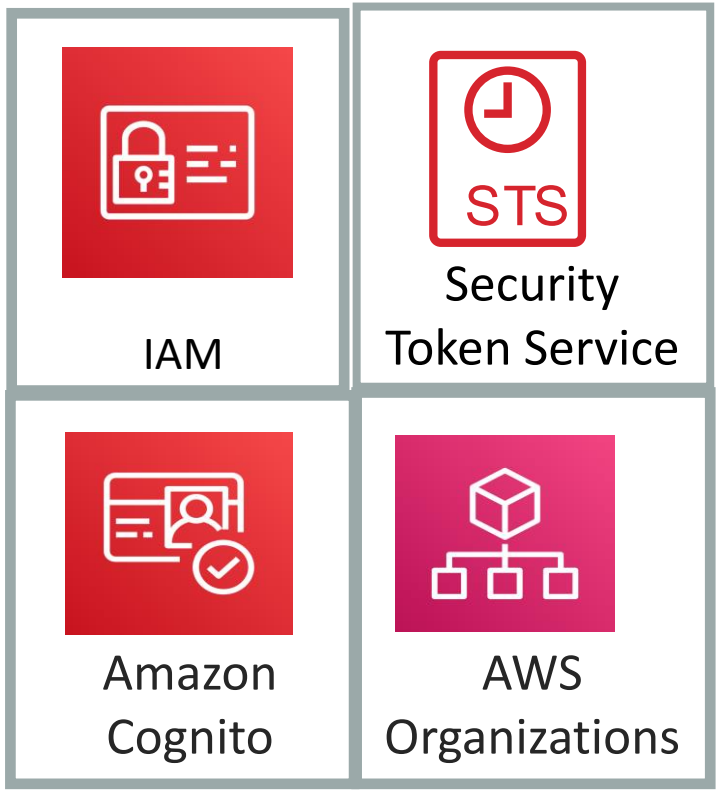
Securing User and Application Access

Module overview

Sections

1. Account users and IAM
2. Organizing users
3. Federating users
4. Multiple accounts

Services we learn



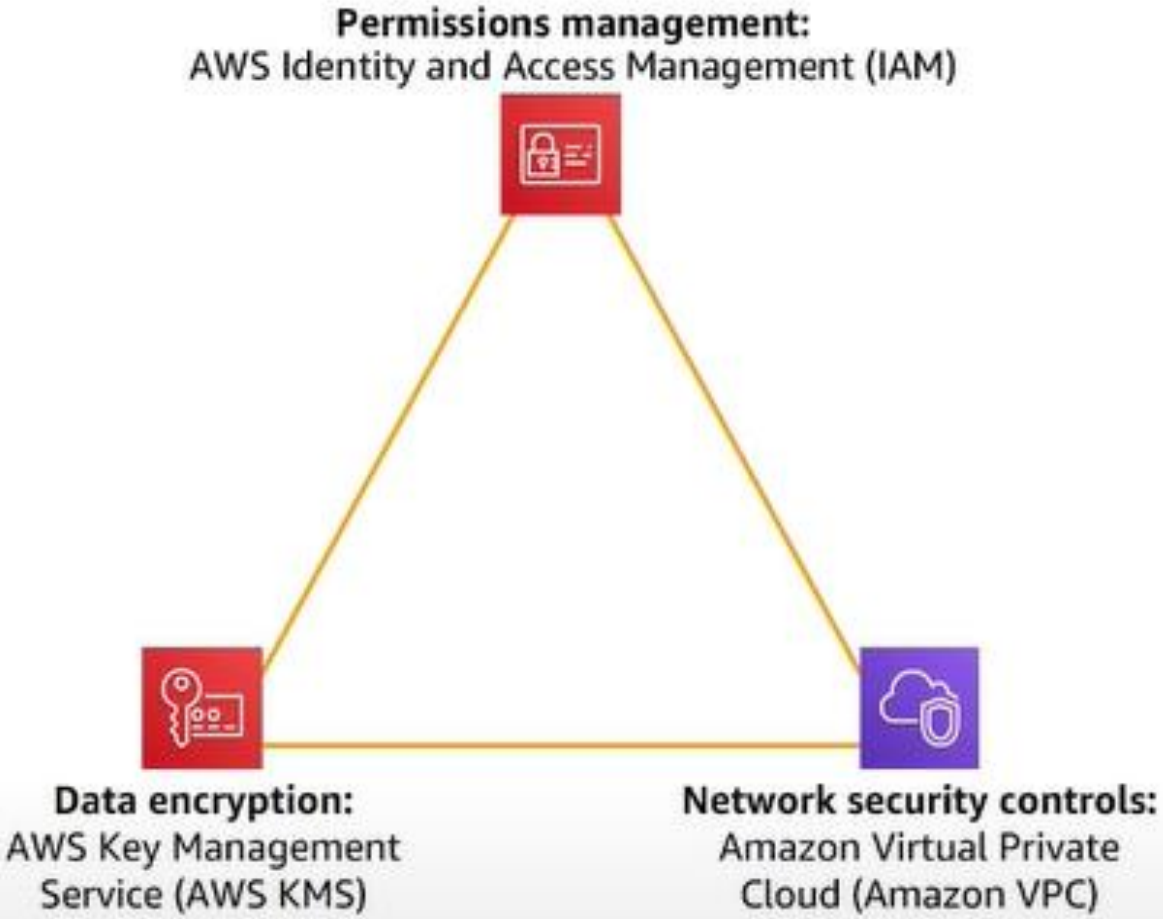
Module objectives

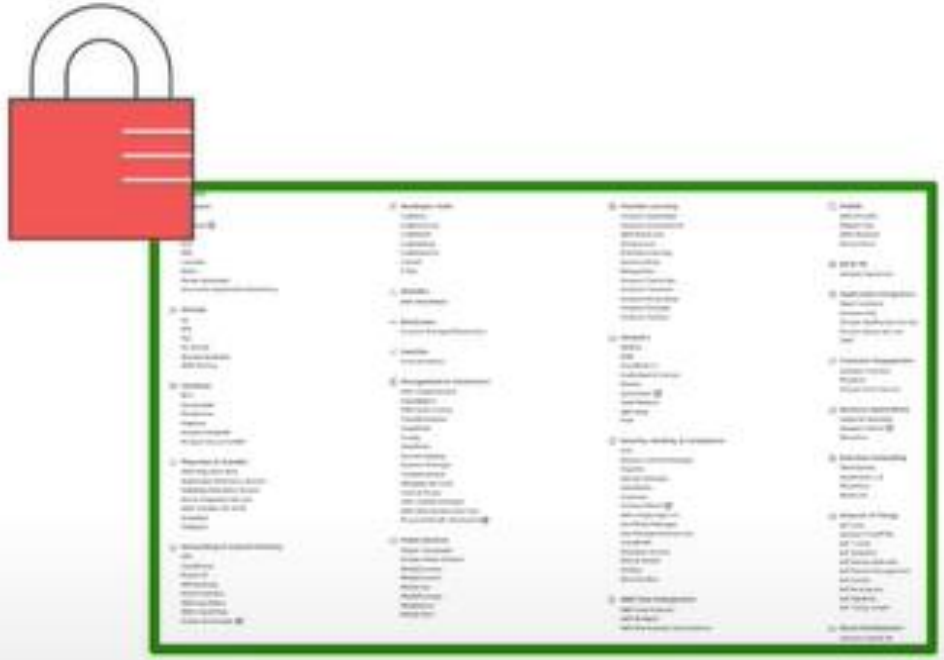
At the end of this module, you should be able to:

- Purpose of AWS Identity and Access Management (IAM) users, groups, and roles
- Describe how to allow user federation within an architecture to increase security
- Recognize how AWS Organizations service control policies (SCPs) increase security within an architecture
- Describe how to manage multiple AWS accounts
- Configure IAM users

Section 1: Account users and IAM

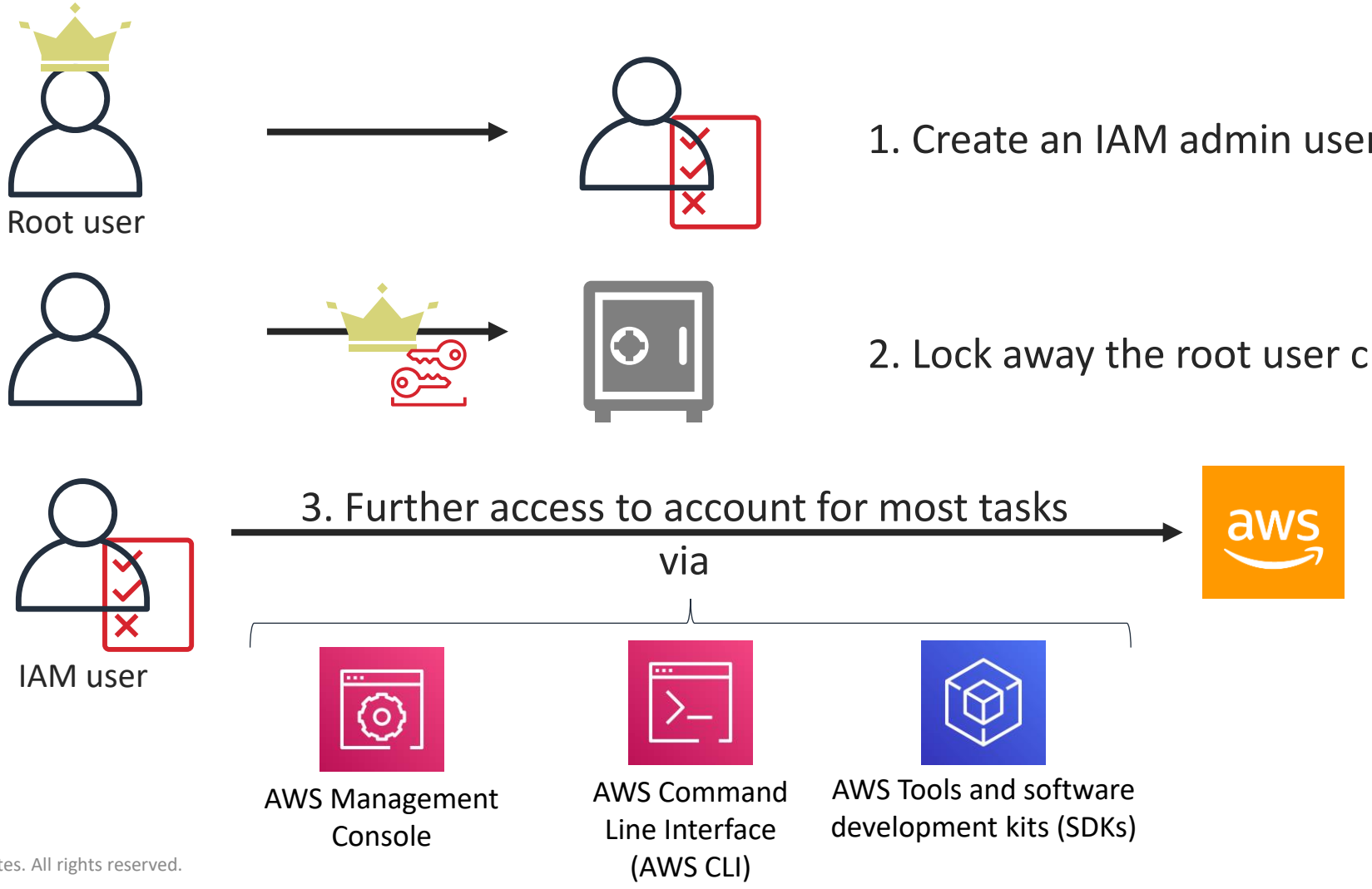
Infrastructure Encryption via IAM service





Secure the root account

The account root user has a large amount of power. Recommended security steps:



AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM)

- Securely control individual and group access to your AWS resources
- Integrates with other AWS services
- Federated identity management
- Granular permissions
- Support for multi-factor authentication

IAM components: Review

IAM user

Defined in your AWS account. Use credentials to authenticate programmatically or via the AWS Management Console.

IAM group

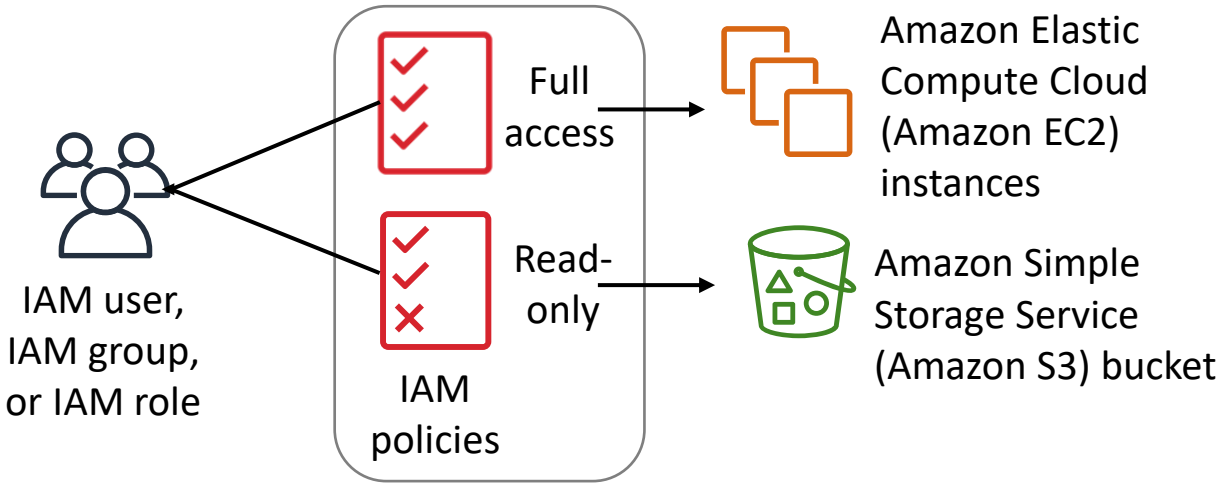
A collection of IAM users that are granted identical authorization.

IAM policy

Defines which resources can be accessed and the level of access to each resource.

IAM role

Mechanism to grant temporary access for making AWS service requests. *Assumable* by a person, application, or service.



Key Points

- How many maximum users can be created in an AWS account?
  - 5000
- How many maximum groups can be created in an AWS account?
  - 100
- Maximum access keys can be assigned to an IAM user
  - 2
- An IAM user cannot be a member of more than 10 groups.
- AWS supports 6 types of policies.
- How many maximum customer-managed policies can be created in an AWS account
  - 1000
- Maximum roles can be created in an AWS account
  - 500

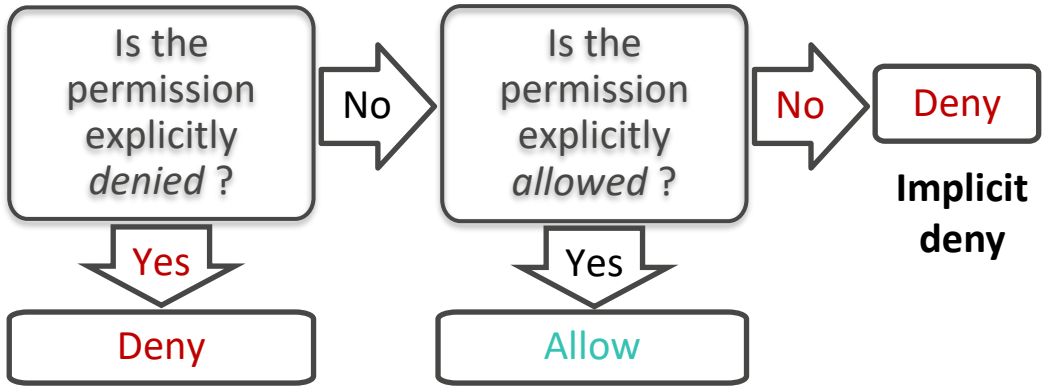
IAM permissions

IAM policy

Permissions are specified in an IAM policy:

- A document formatted in JavaScript Object Notation (JSON)
- It defines which resources and operations are allowed
- Best practice – follow the principle of least privilege
- Two types of policies –
  - Identity-based: Attach to an IAM principal
  - Resource-based: Attach to an AWS resource

How IAM determines permissions at the time of request:



Identity-based versus resource-based policies

Identity-based policies

- Attached to a user, group, or role
- Types of policies
  - AWS managed
  - Customer managed
  - Inline

Resource-based policies

- Attached to AWS resources
  - Example: Attach to an Amazon S3 bucket
- Always an inline policy

IAM policy document structure

```
{
   "Version": "2012-10-17",
   "Statement":[{
      "Effect": "effect",
      "Action": "action",
      "Resource": "arn",
      "Condition":{
         "condition":{
            "key": "value"
         }
      }
   }]
}
```

- Effect: Effect can be either *Allow* or *Deny*
- Action: Type of access that is allowed or denied

```
"Action": "s3:GetObject"
```

- Resource: Resources that the action will act on

```
"Resource": "arn:aws:sqs:us-west-2:123456789012:queue1"
```

- **Condition**: Conditions that must be met for the rule to apply

```
"Condition" : {
   "StringEquals" : {
         "aws:username" : "johndoe"
   }
}
```

ARNs and wildcards

- Resources are identified by using Amazon Resource Name (ARN) format
  - Syntax – `arn:`*`partition`*`:`*`service`*`:`*`region`*`:`*`account`*`:`*`resource`*
  - Example – `"Resource": "arn:aws:iam::123456789012:user/mmajor"`

- You can use a wildcard (*) to give access to all actions for a specific AWS service
  - Examples –
    - `"Action": "s3:*"`
    - `"Action": "iam:*AccessKey*"`

IAM policy example

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":["DynamoDB:*","s3:*"],
    "Resource":[
      "arn:aws:dynamodb:region:account-number-without-hyphens:table/table-name",
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    },
    {
    "Effect":"Deny",
    "Action":["dynamodb:*","s3:*"],
    "NotResource":["arn:aws:dynamodb:region:account-number-without-hyphens:table/table-name",
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    }
  ]
}
```

Explicit allow gives users access to a specific DynamoDB table and…

…Amazon S3 buckets.

Explicit deny ensures that the users cannot use any other AWS actions or resources other than that table and those buckets.

An explicit deny statement **takes precedence** over an allow statement.

Activity: Examining IAM policies

Photo by Pixabay from Pexels.

Activity: IAM policy analysis (1 of 3)

Consider this IAM policy, then answer the questions as they are presented.

```
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "iam:Get*",
         "iam:List*"
      ],
      "Resource": "*"
   }
}
```

1. Which AWS service does this policy grant you access to?
   - ANSWER: The IAM service.
2. Does it allow you to create an IAM user, group, policy, or role?
   - ANSWER: No. The access is limited to *get* and *list* requests. It effectively grants read-only permissions.
3. Name at least three specific actions that the iam:Get* action allows.
   - ANSWER: iam:Get* allows many specific actions, including GetGroup, GetPolicy, GetRole, and others.

Activity: IAM policy analysis (2 of 3)

Consider this IAM policy, then answer the questions as they are presented.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ec2:TerminateInstances"],
            "Resource": ["*"]
        },
        {
            "Effect": "Deny",
            "Action": ["ec2:TerminateInstances"],
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            },
            "Resource": ["*"]
        }
    ]
}
```

1. Does the policy allow you to terminate any EC2 instance at any time without conditions?
   - ANSWER: No. The first statement object allows it. However, the second statement object applies a condition.
2. Are you allowed to make the terminate instance call from anywhere?
   - ANSWER: No. You can only make the request from one of the two IP address ranges that are specified in *aws:SourceIp*.
3. Can you terminate instances if you make the call from a server that has an assigned IP address of *192.0.2.243*?
   - ANSWER: Yes, because the 192.0.2.0/24 Classless Inter-Domain Routing (CIDR) IP address range includes IP addresses 192.0.2.0 through 192.0.2.255. A resource like the CIDR to IP Range tool can be used to calculate the range of a CIDR block.

Activity: IAM Policy analysis (3 of 3)

Consider this IAM policy, then answer the questions as they are presented.

```
{
  "Version": "2012-10-17",
  "Statement":[{
      "Condition": {
        "StringNotEquals": {
           "ec2:InstanceType": [
              "t2.micro",
              "t2.small"
           ]
        }
      },
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Action": [
         "ec2:RunInstances",
         "ec2:StartInstances"
      ],
     "Effect": "Deny"
}
```

1. What actions does the policy allow?
   - ANSWER: It does not allow you to do anything (the effect is to *Deny*).
2. Say that the policy included an additional statement object, like this example:

   ```
   {
        "Effect": "Allow",
        "Action": "ec2:*"
   }
   ```

   How would the policy restrict the access granted to you by this additional statement?
   - ANSWER: You would have full Amazon EC2 service access. However you would only be allowed to launch or start EC2 instances of instance type t2.micro or t2.small.
3. If the policy included both the statement on the left and the statement in question 2, could you terminate an m3.xlarge instance that existed in the account?
   - ANSWER: Yes.

Section 2: Organizing users

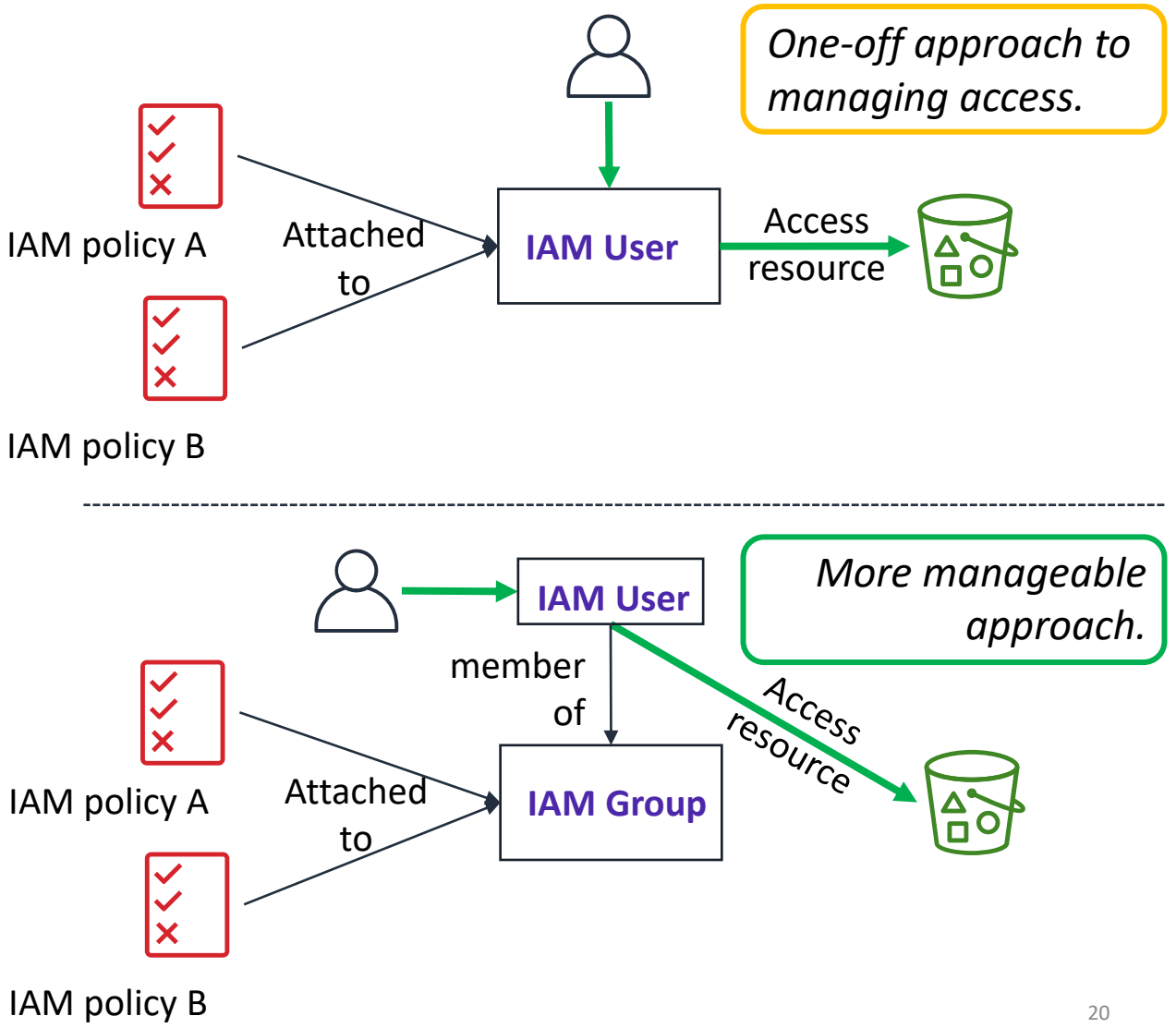
IAM groups

Use IAM groups to grant the same access rights to multiple users.

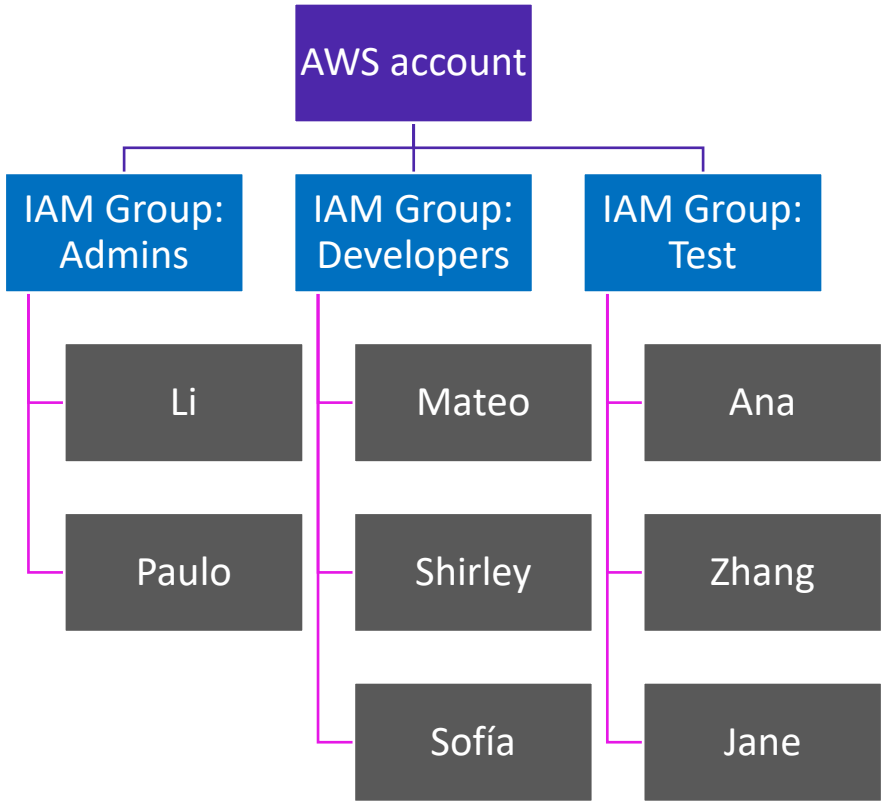
- All users in the group inherit the permissions assigned to the group
  - Makes it easier to manage access across multiple users

Tip: Combine approaches for fine-grained individual access

- Add the user to a group to apply standard access based on job function
- Optionally attach an additional policy to the user for needed exceptions



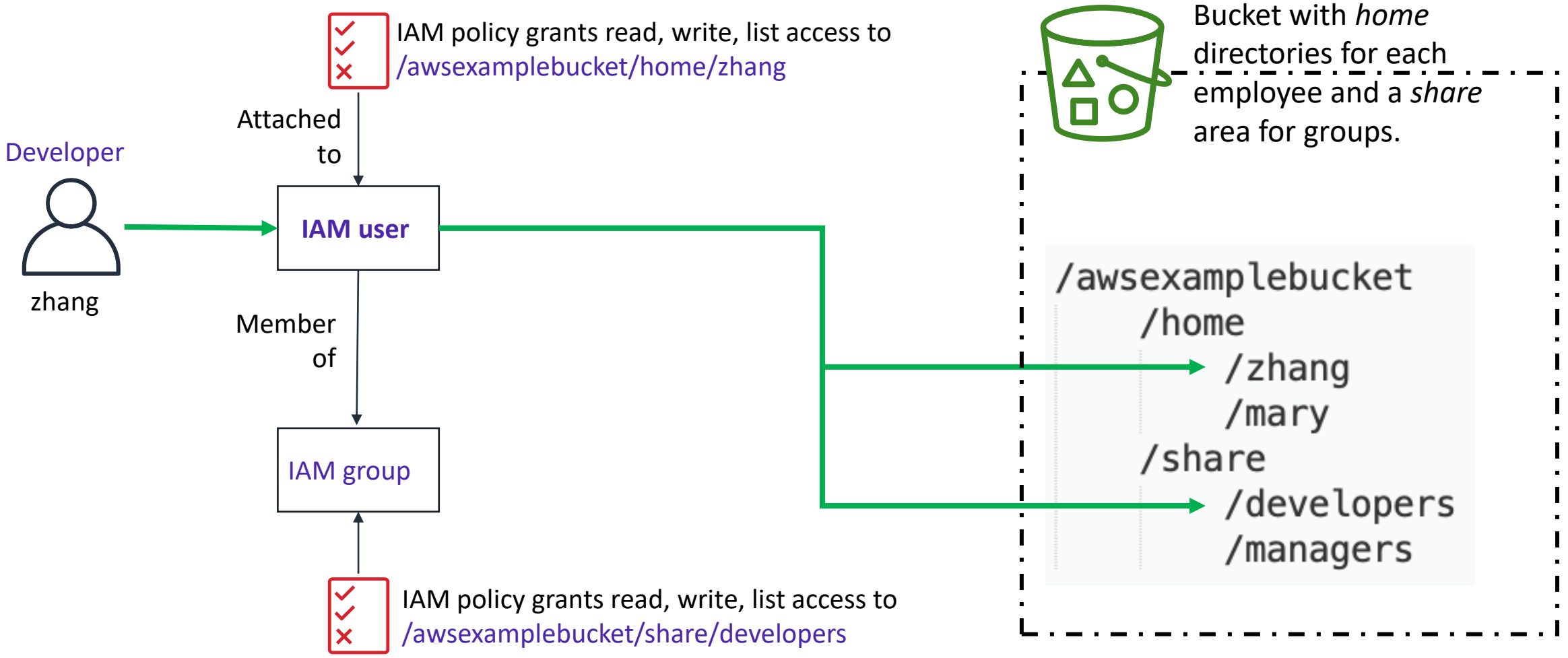
Example IAM groups



Tip: Create groups that reflect job functions

- If a new developer is hired, add them to the *Developer* group
  - Immediately inherit the same access granted to other developers
- If Ana takes on the new role of developer –
  - Remove her from the *Test* group
  - Add her to the *Developer* group
- Users can belong to more than one group
  - However the most restrictive policy will then apply

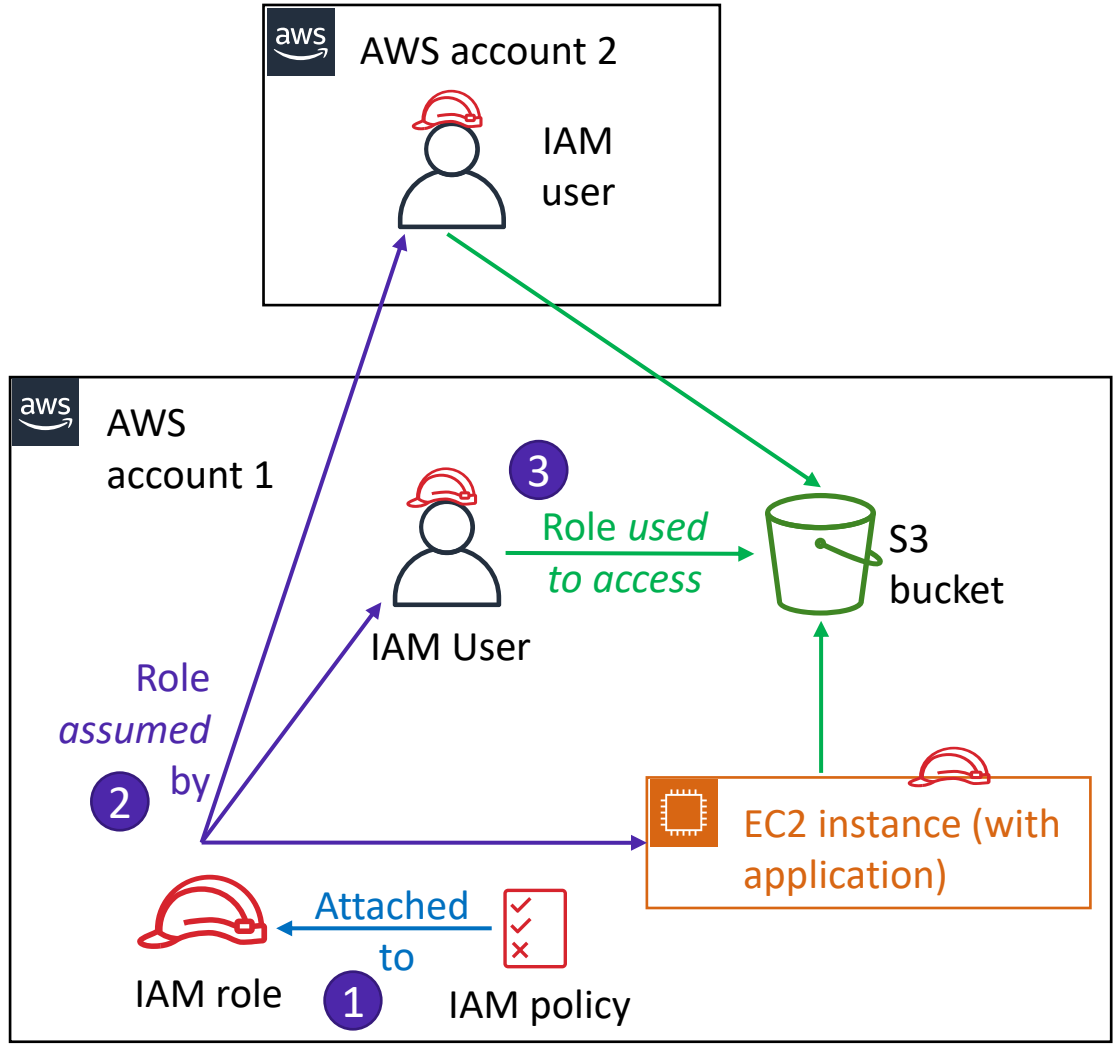
Use case for IAM with Amazon S3



Section 3: Federating users

IAM roles

- IAM role characteristics
  - Provides *temporary* security credentials
  - Is not uniquely associated with one person
  - Is *assumable* by a person, application, or service
  - Is often used to delegate access
- Use cases
  - Provide AWS resources with access to AWS services
  - Provide access to externally authenticated users
  - Provide access to third parties
  - Switch roles to access resources in –
    - Your AWS account
    - Any other AWS account (cross-account access)



Grant permissions to assume a role

AWS Security Token Service (AWS STS)

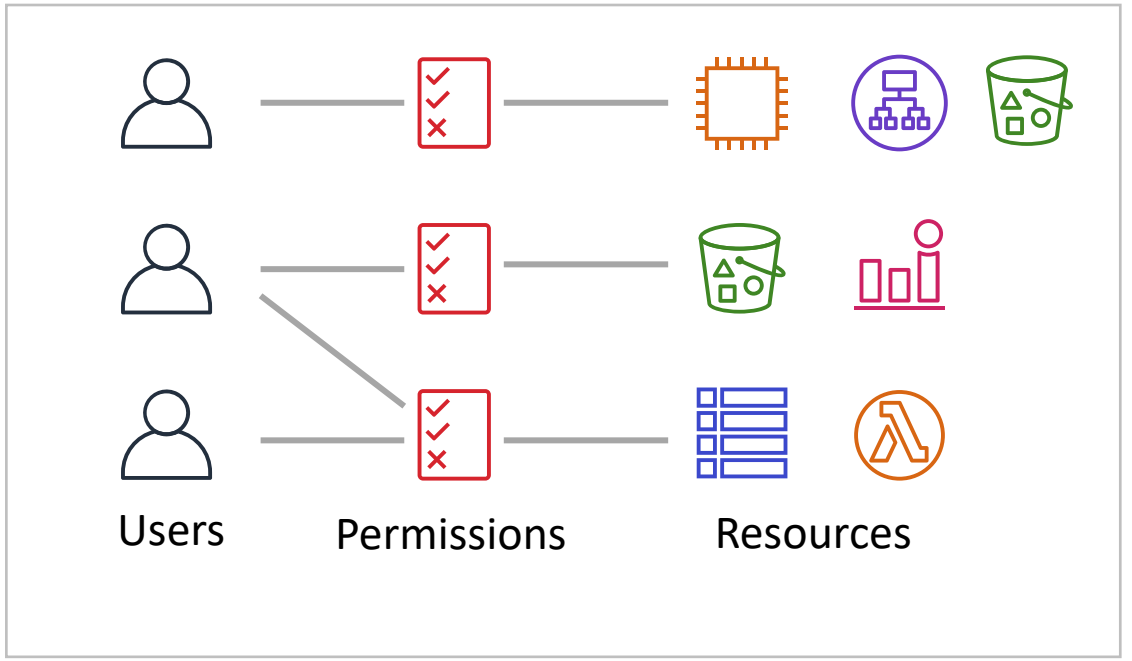
- For an IAM user, application, or service to assume a role, you must grant permissions to switch to the role
- AWS Security Token Service (AWS STS)
  - Web service that enables you to request temporary, limited-privilege credentials
  - Credentials can be used by IAM users or for users that you authenticate (federated users)
- Example policy – Allows an IAM user to assume a role

```
{
   "Version": "2012-10-17",
   "Statement": {
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::123456789012:role/Test*"
   }
}
```

Role-based access control (RBAC)

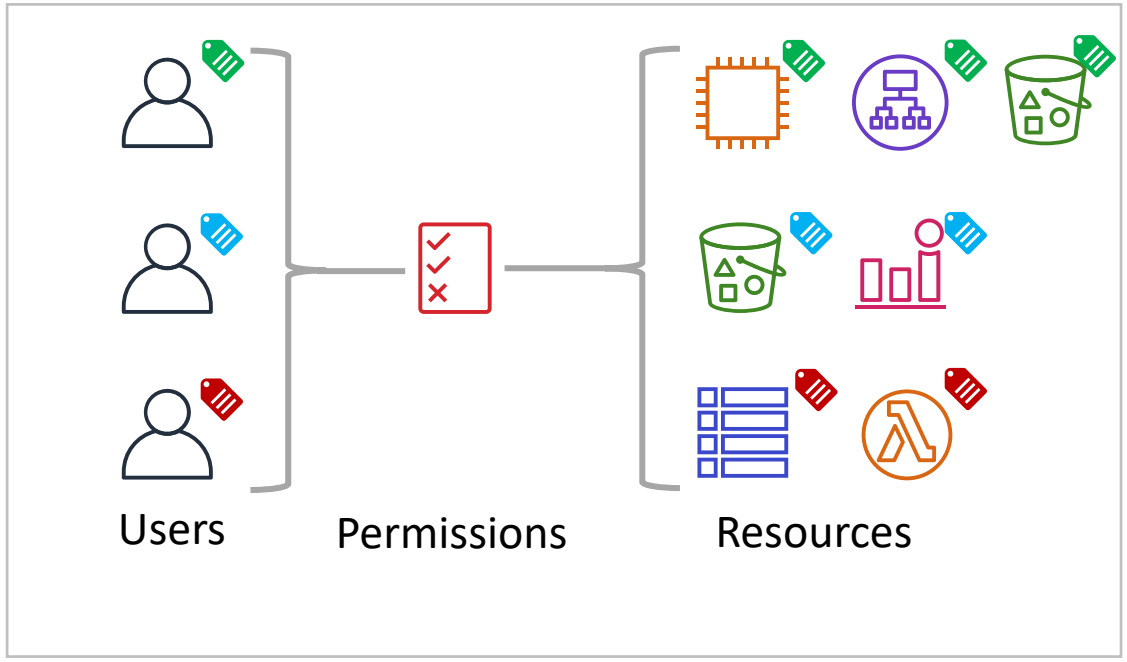
Traditional approach to access control:

- Grant users specific permissions based on job function (such as database administrator)
- Create a distinct IAM role for each permission combination
- Update permissions by adding access for each new resource (it can become time-consuming to keep updating policies)



Attribute-based access control (ABAC)

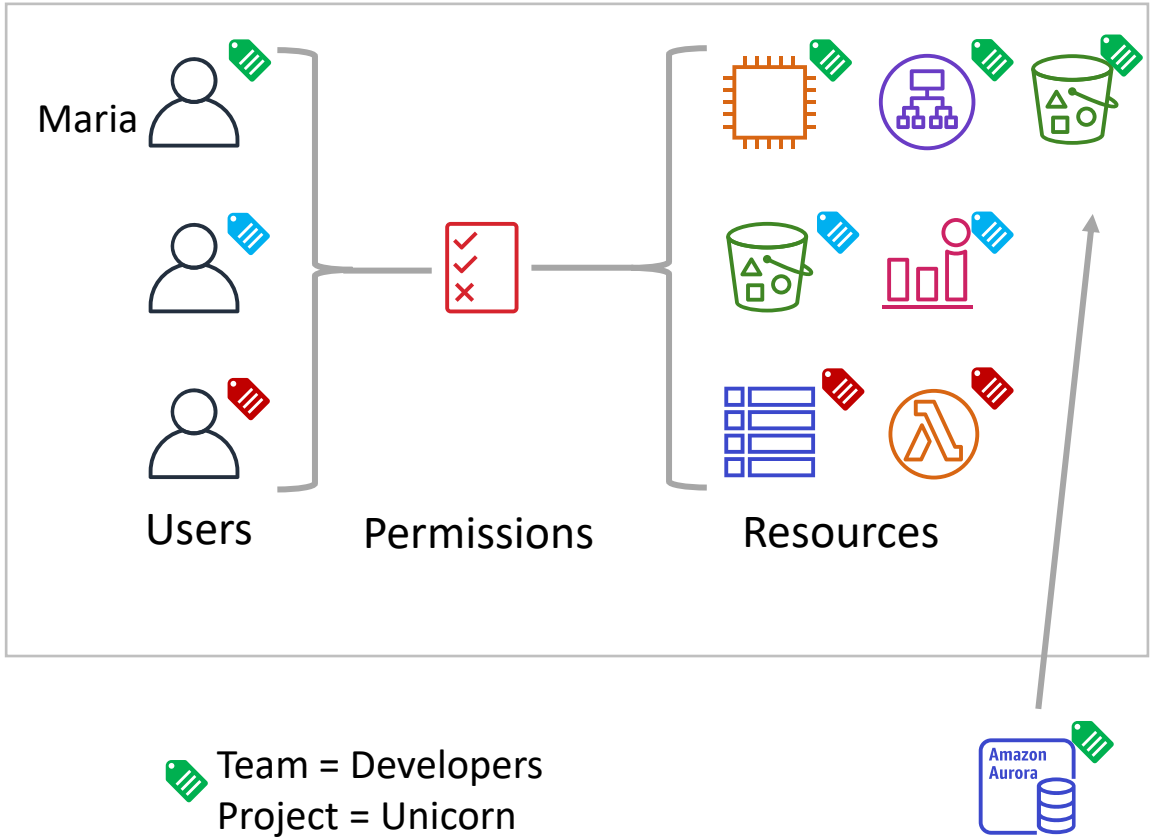
- Highly scalable approach to access control
  - Attributes are a key or a key-value pair, such as a tag
  - Example attributes –
    - Team = Developers
    - Project = Unicorn
- Permissions (policy) rules are easier to maintain with ABAC than with RBAC
- Benefits
  - Permissions automatically apply, based on attributes
  - Granular permissions are possible *without* a permissions update for every new user or resource
  - Fully auditable



Applying ABAC to your organization

How to apply ABAC to your organization:

1. Set access control attributes on identities
2. Require attributes for new resources
3. Configure permissions based on attributes
4. Test
   a) Create new resources
   b) Verify that permissions automatically apply



Externally authenticated users

Identity federation

- User authentication completed by a system that is external to the AWS account
  - Example: corporate directory
- It provides a way to allow access through existing identities, without creating IAM users

Identity federation options

1. AWS STS
   - Public identity service providers (IdPs)
   - Custom identity broker application
2. Security Assertion Markup Language (SAML)
3. Amazon Cognito

IdP authentication overview

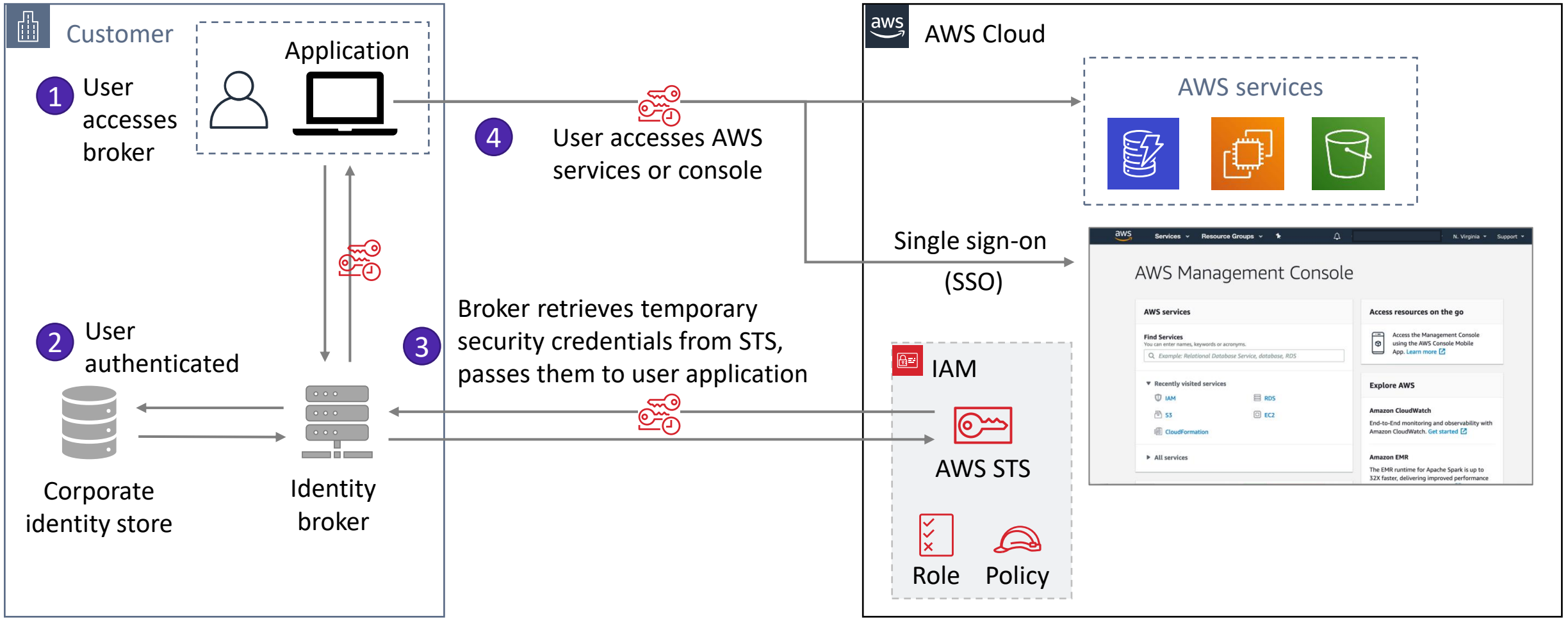
User accesses identity broker via application

Identity broker authenticates user

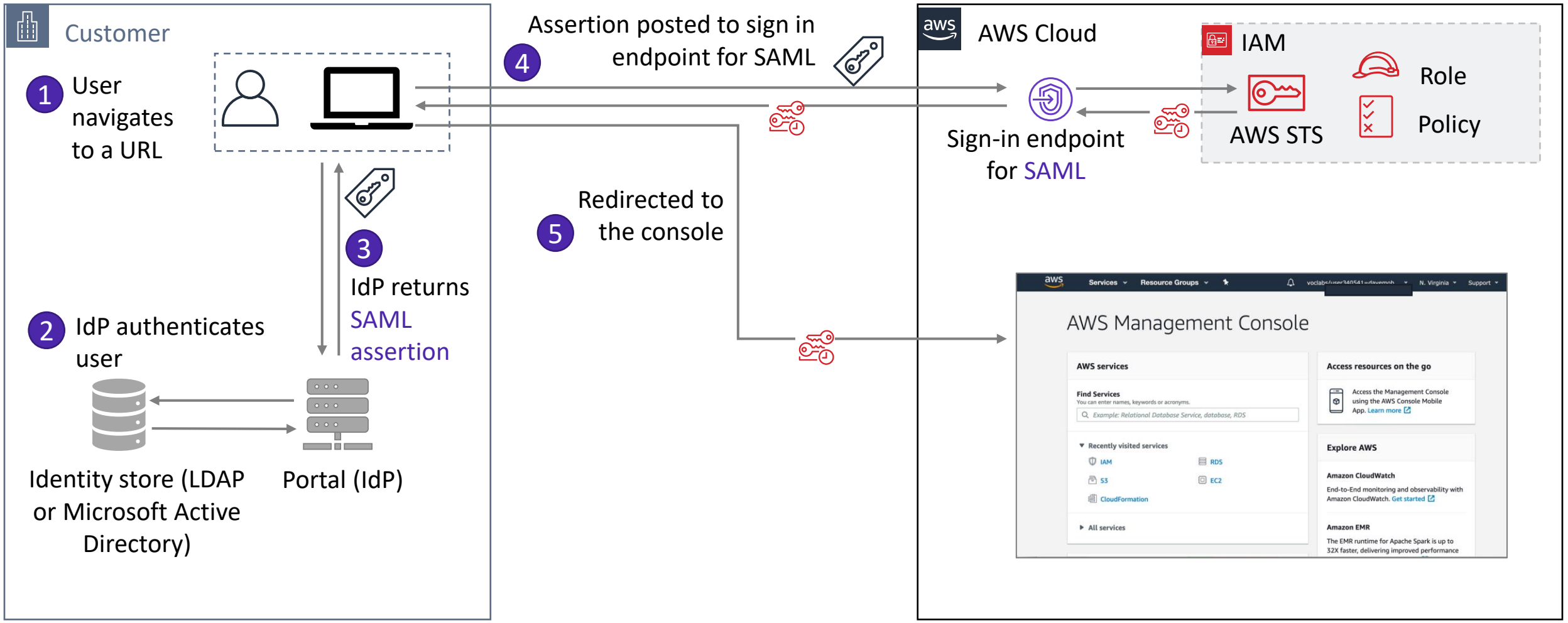
Requests temporary credentials from AWS STS

Temporary credentials returned to application

Identity federation with an identity broker



Identity federation using SAML



Amazon Cognito

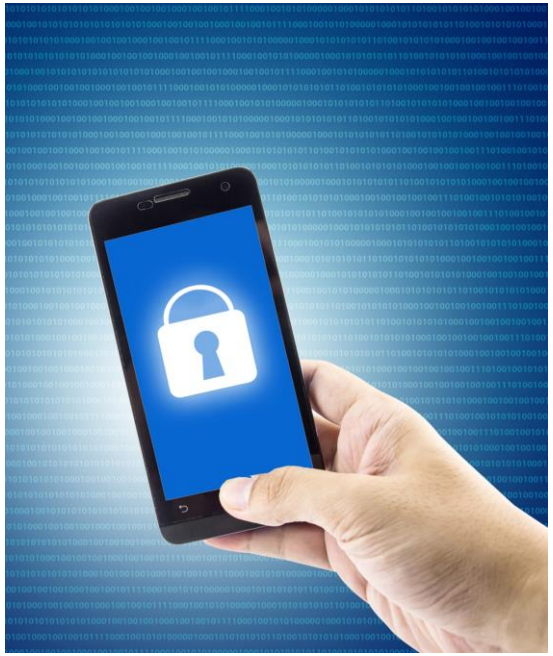
Amazon Cognito

Amazon Cognito is a fully managed service.

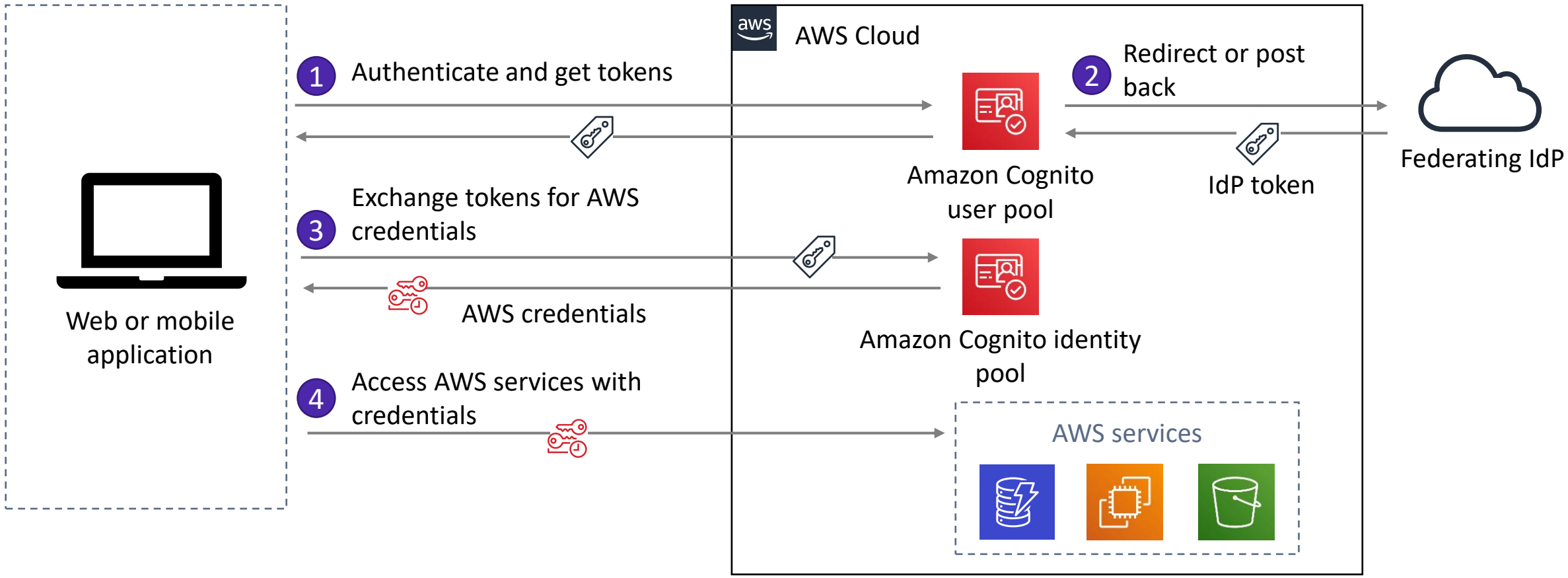
- It provides authentication, authorization, and user management for web and mobile applications

- Amazon Cognito provides web identity federation
- Federated identities
  - Users sign in with social identity providers (Amazon, Facebook, Google) or with SAML

- User pools
  - You can maintain a directory with user profiles authentication tokens



Amazon Cognito example



Section 4: Multiple accounts

One account or multiple accounts?

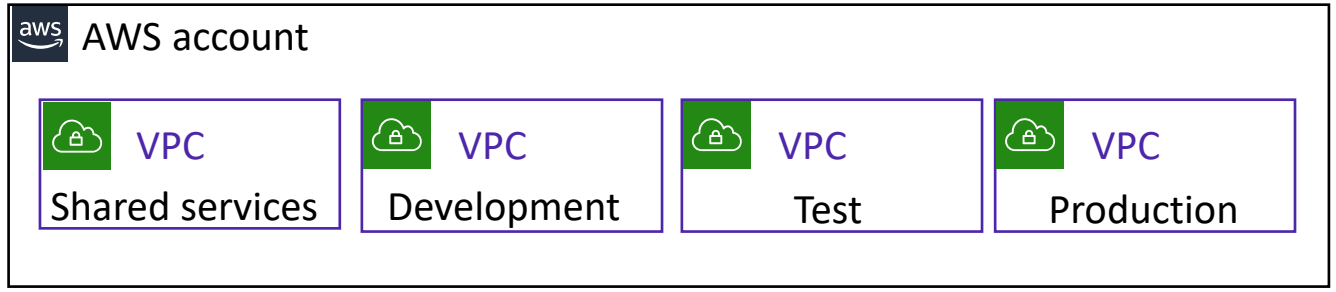
Two architectural patterns

- Most organizations choose to create multiple accounts

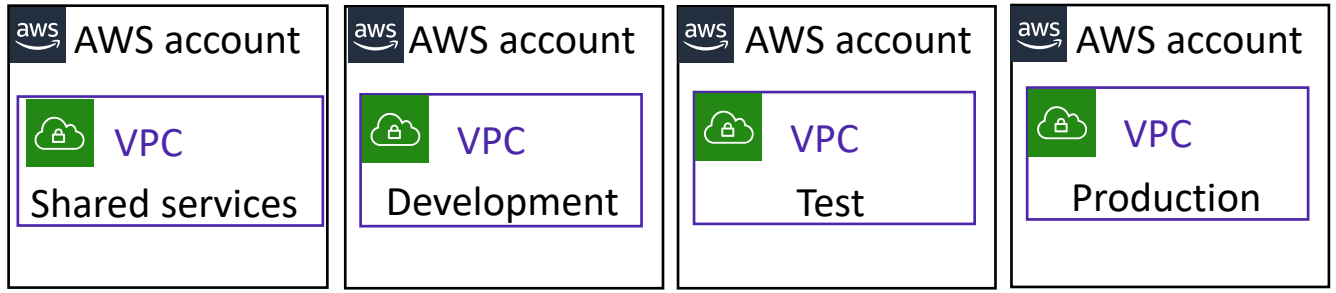
Advantages of multiple accounts

- Isolate business units or departments
- Isolate development, test, and production environments
- Isolate auditing data, recovery data
- Separate accounts for regulated workloads
- Easier to trigger cost alerts for each business unit's consumption

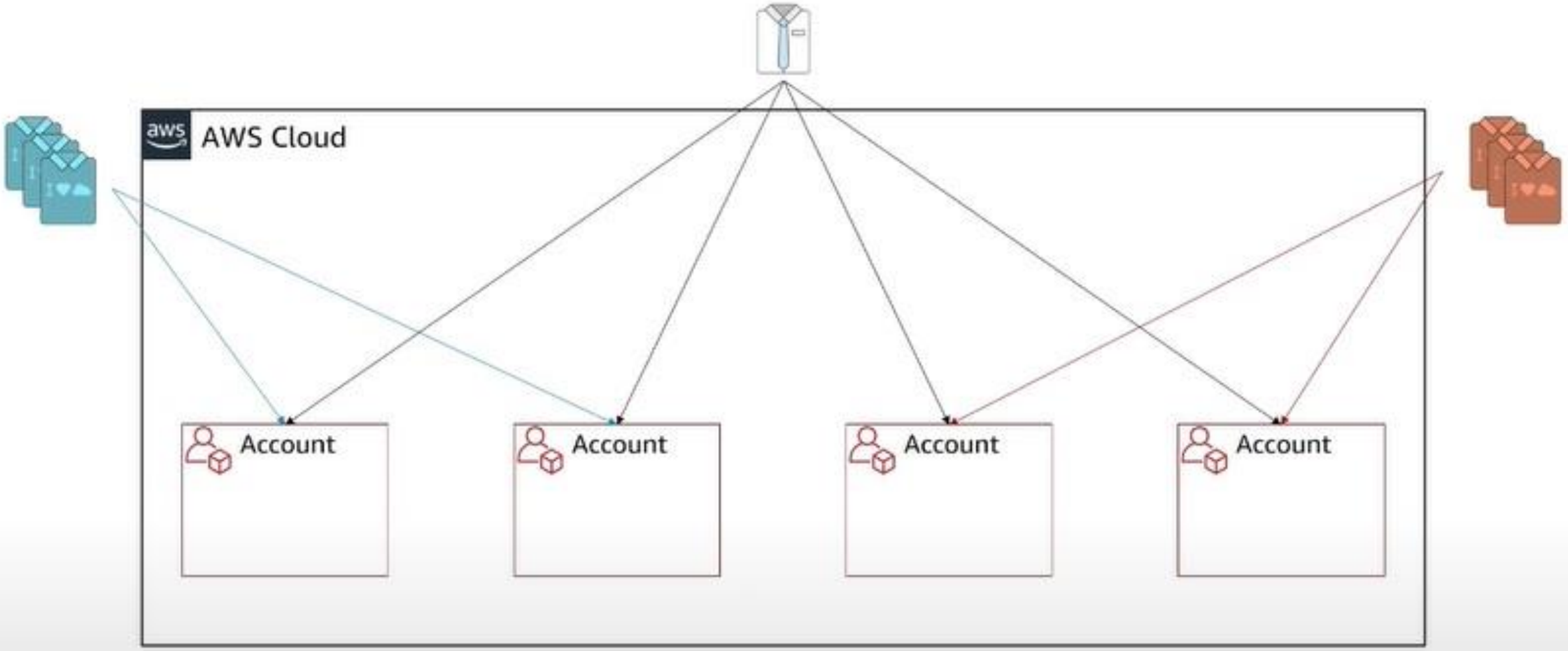
Multiple VPCs in a single account
architectural pattern



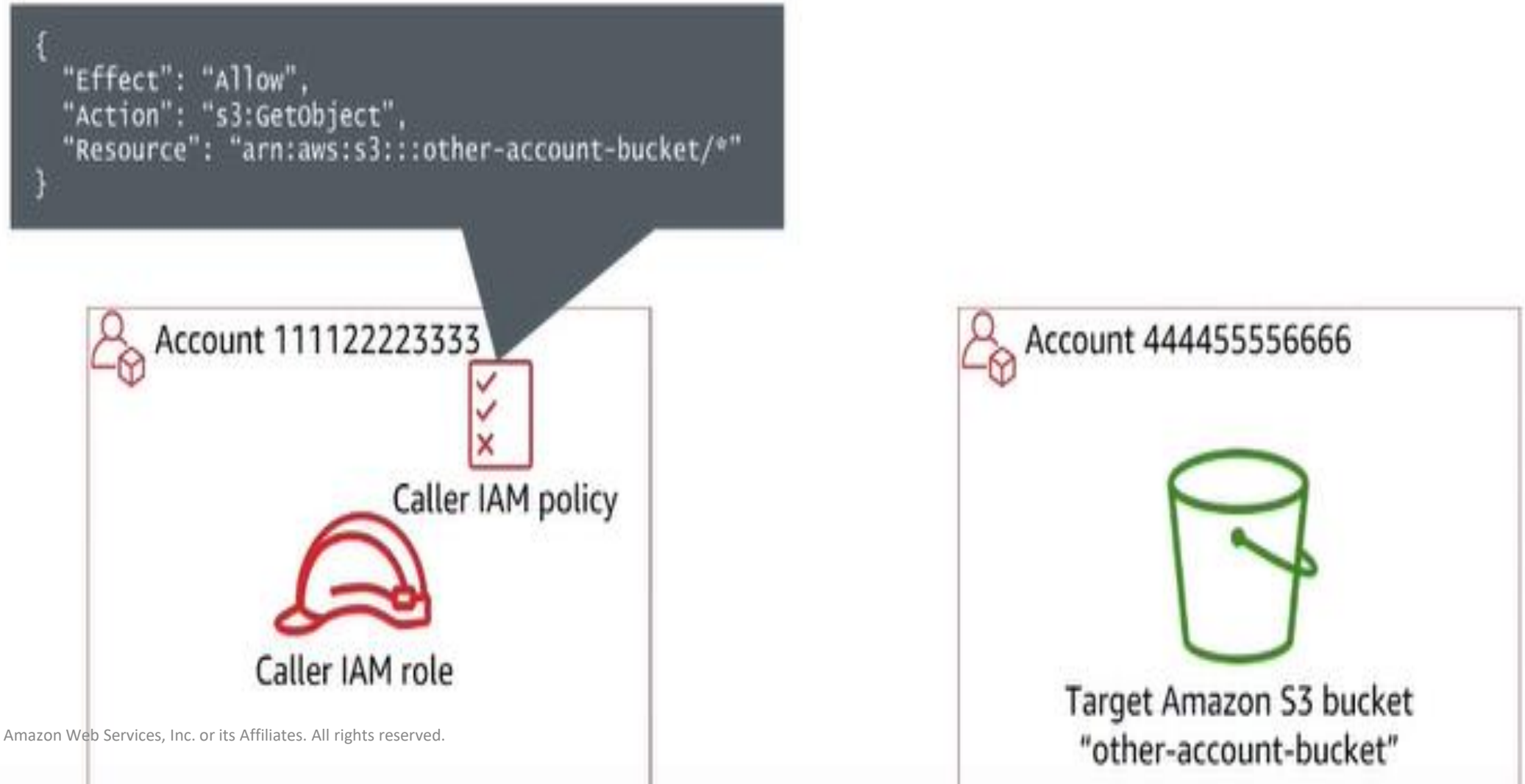
Multiple accounts, a VPC in each account
architectural pattern



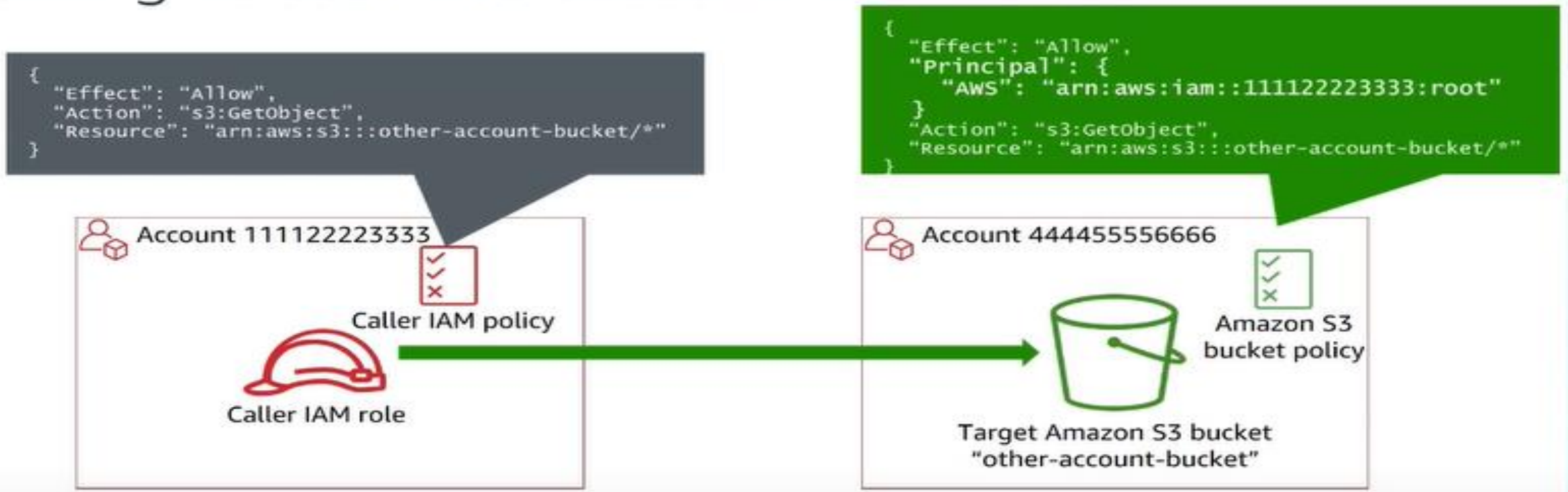
IAM in an AWS enterprise environment



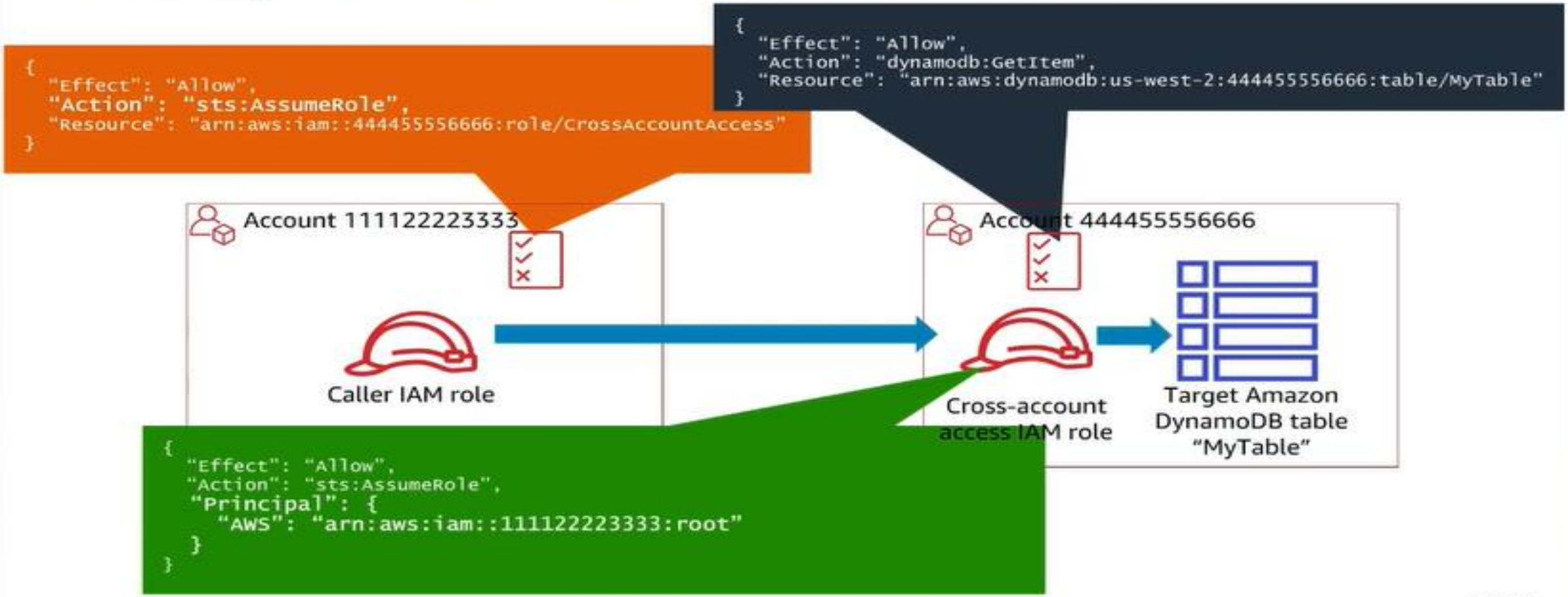
Working across AWS account boundaries



Working across AWS account boundaries



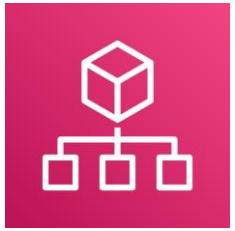
Working across AWS account boundaries



Challenges for managing multiple accounts

- Security management across accounts
  - IAM policy replication
- Creating new accounts
  - Involves many manual processes
- Billing consolidation
- Centralized governance is needed to ensure consistency

Manage multiple accounts with AWS Organizations



AWS Organizations

Centrally manage and enforce policies across multiple AWS accounts.

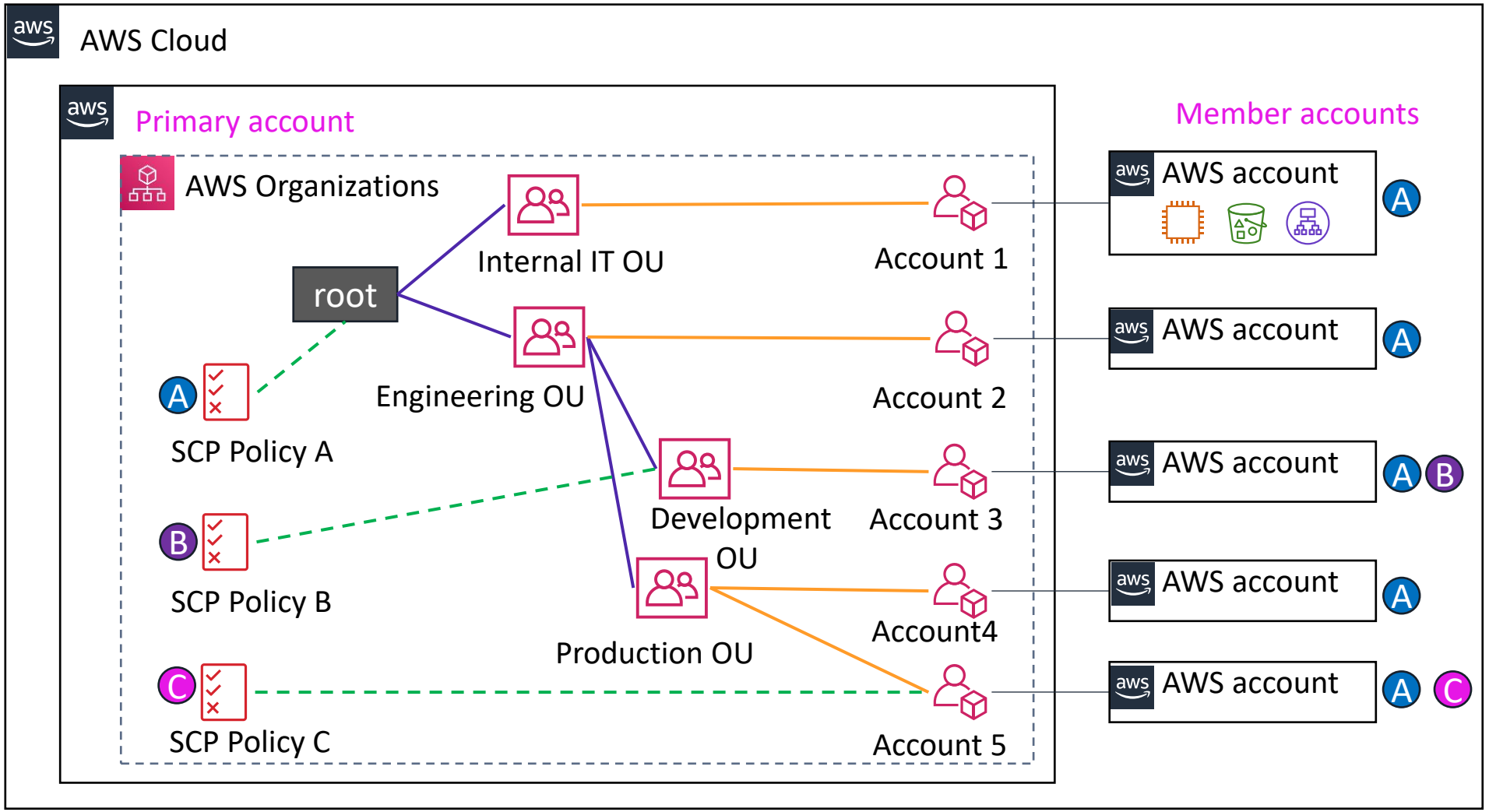
- Group-based account management
- Policy-based access to AWS services
- Automated account creation and management
- Consolidated billing
- API-based

AWS Organizations: Illustrated

In the AWS Organizations primary account:

1. Create a hierarchy of organizational units (OUs)
2. Assign accounts to OUs as member accounts
3. Define service control policies (SCPs) that apply permissions restrictions to specific member accounts
4. Attach the SPCs to root, OUs, or accounts

Which accounts does each SCP apply to?



Example uses of SCPs

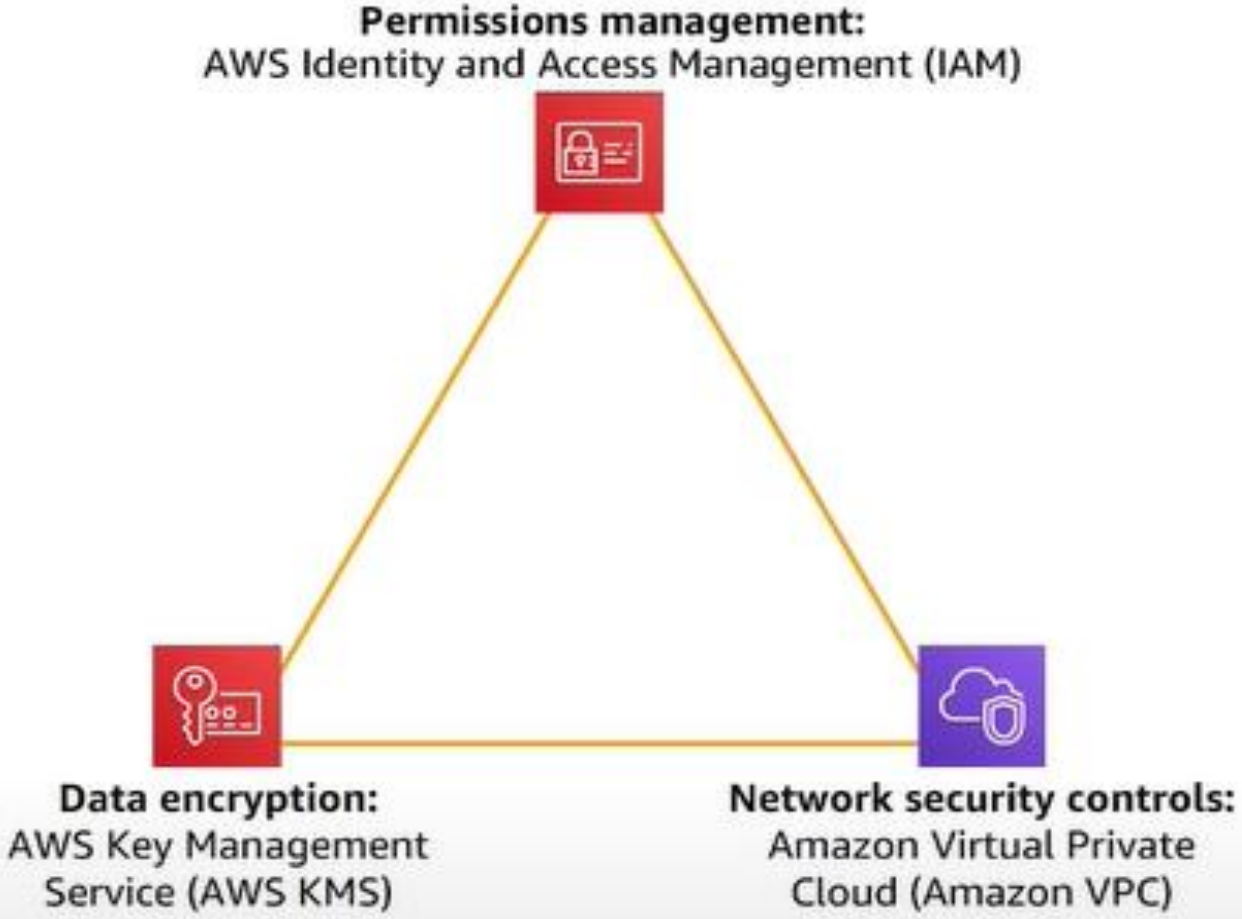
- Characteristics of service control policies (SCPs)
  - They enable you to control which services are accessible to IAM users in member accounts
  - SCPs cannot be overridden by the local administrator
  - IAM policies that are defined in individual accounts still apply

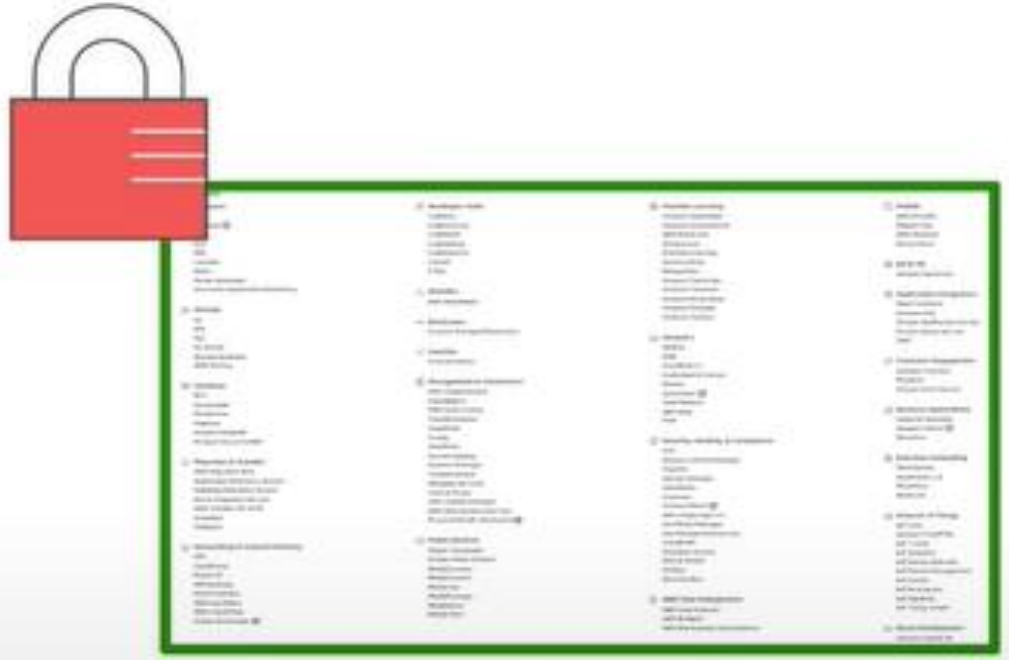
- Example uses of SCPs
  - Create a policy that *blocks* service access or specific actions

    Example: Deny users from disabling AWS CloudTrail in all member accounts
  - Create a policy that *allows* full access to specific services

    Example: Allow full access to Amazon EC2 and CloudWatch
  - Create a policy that *enforces the tagging* of resources

Data Encryption via AWS KMS





AWS KMS

- **What it is:** AWS-managed encryption and decryption service
- **Why it matters to you:** Many data-handling AWS services offer simple AWS KMS integrations; if you know how to use AWS KMS, you can protect your data at rest simply and with no management overhead
- **What builders need to know**
  - The basics of how to use an AWS KMS key
  - The AWS KMS integrations offered by many AWS data-handling services
  - How to use IAM to control access to keys

The mechanics of an AWS KMS key

AWS KMS key

For encrypting individual pieces of data (<=4KB)

- KMS.Encrypt("hello world") ➔ AQICAHiwKPHZcwiIv….
- KMS.Decrypt("AQICAHiwKPHZcwiIv….") ➔ "hello world"

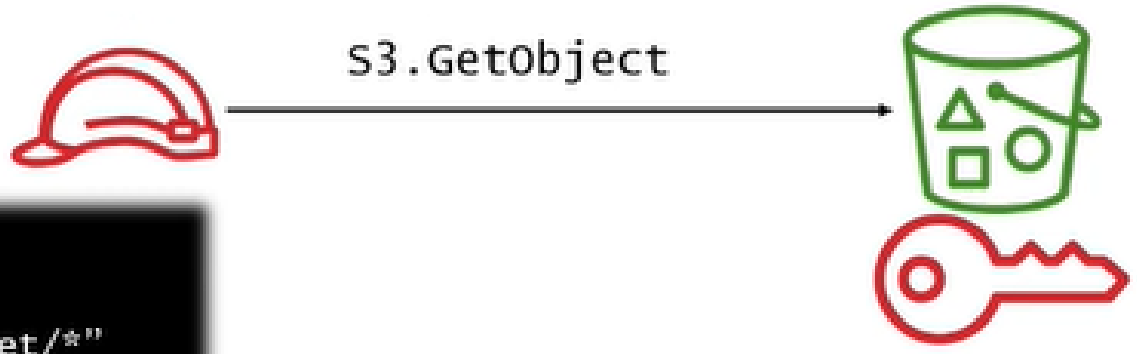
For encrypting application data, use envelope encryption

- KMS.GenerateDataKey ➔ symmetric data key (plaintext and encrypted)
- Use plaintext data key to encrypt your data, then discard
- Store encrypted data key alongside your data
- To decrypt
  - KMS.Decrypt(encryptedDataKey) ➔ plaintextDataKey
  - Then decrypt the data with the plaintext symmetric key

aws

IAM permissions for AWS KMS keys

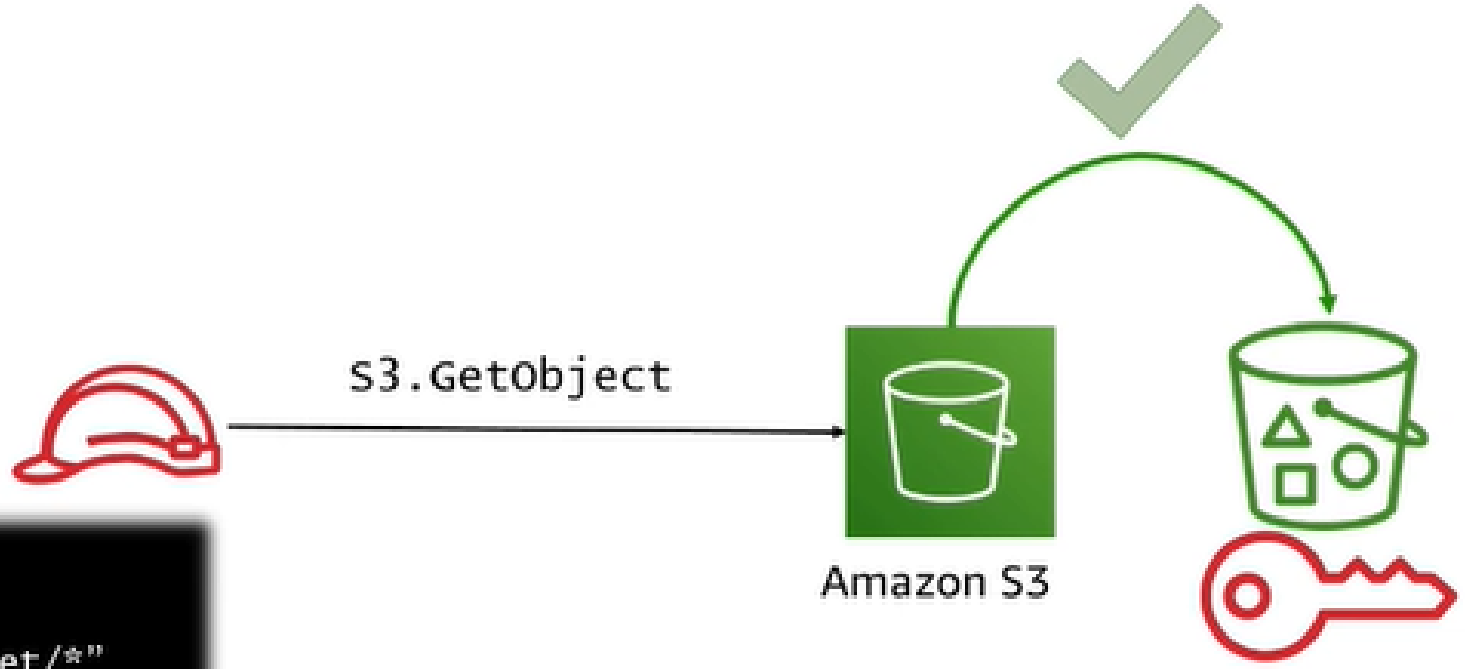
Question: What happens here?




```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::my-bucket/*"
}
```


aws

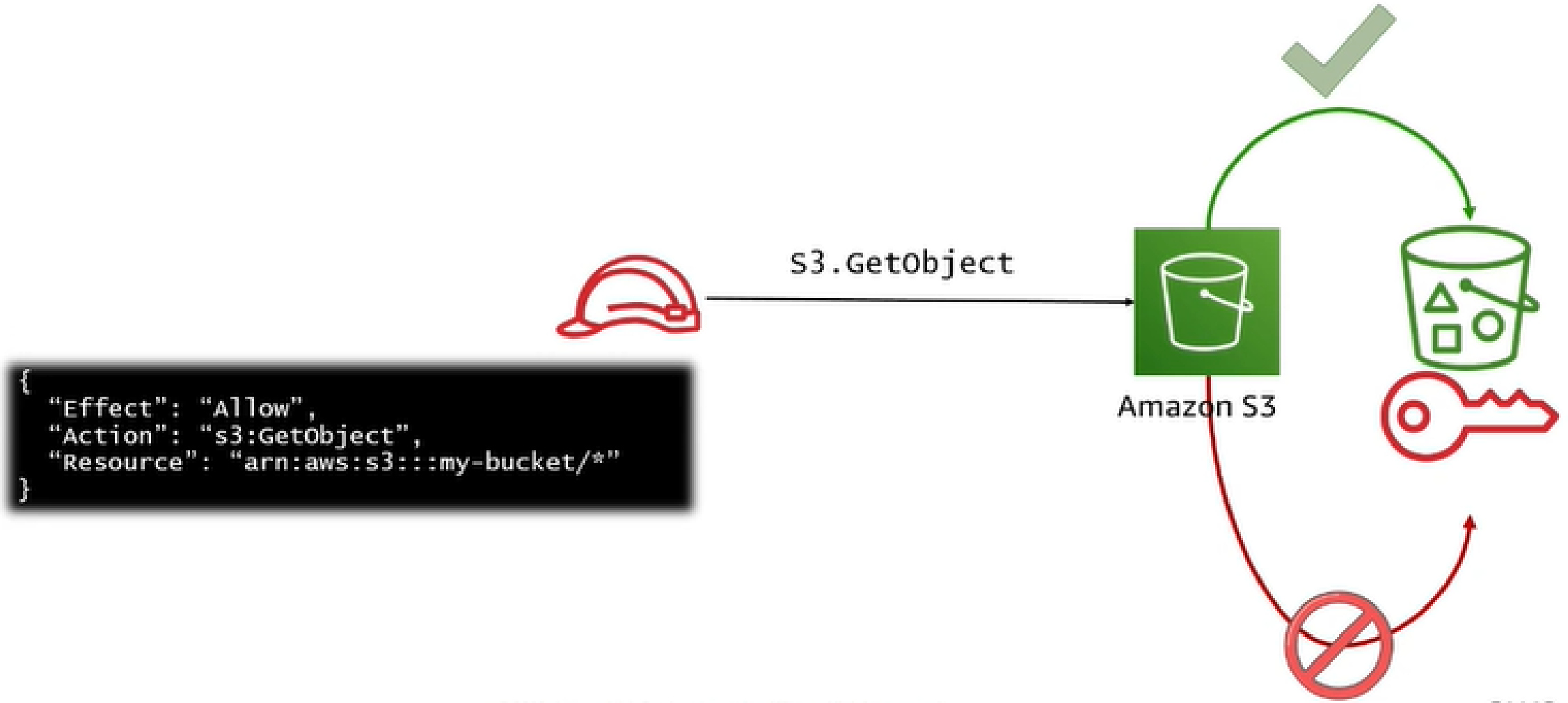
IAM permissions for AWS KMS keys




```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::my-bucket/*"
}
```

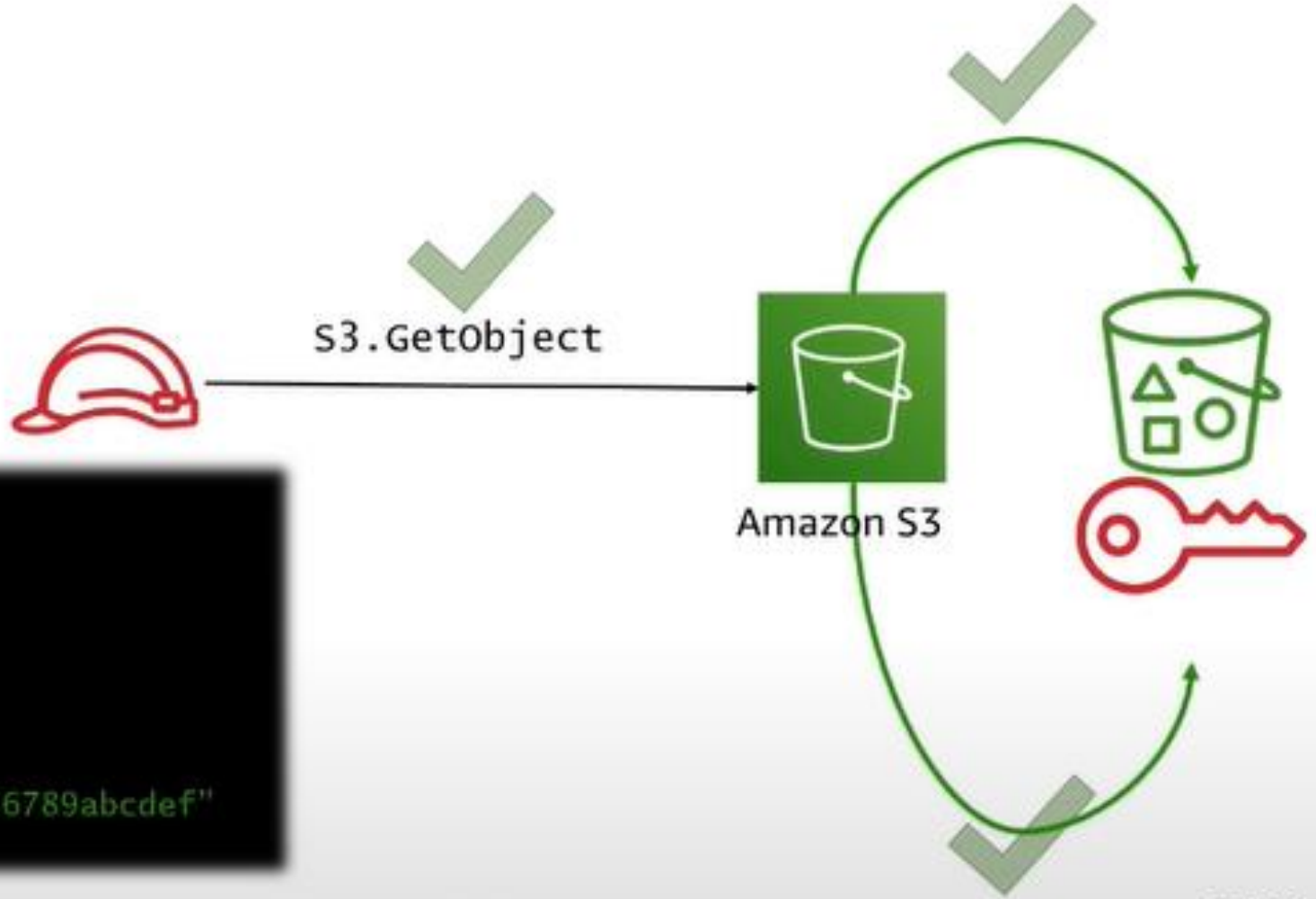

aws

IAM permissions for AWS KMS keys




```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::my-bucket/*"
}
```

IAM permissions for AWS KMS keys




```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::my-bucket/*"
},
{
  "Effect": "Allow",
  "Action": "kms:Decrypt",
  "Resource": "arn:aws:kms:us-east-
2:111122223333:key/01234567-89ab-cdef-0123-456789abcdef"
}
```


aws

Lab Demo

KMS

- Key creation using AWS KMS
- Securing services via the key
  - AWS S3 service
  - AWS DynamoDB service
  - AWS RDS service
  - AWS EC2 service

AWS Organizations

- Associate an account to AWS organization
- Attach accounts to Organizational units (OUs)
- Switch role from one account to another
- Create SCP in AWS organization.