

CSCI 5902 Adv. Cloud Architecting
Fall 2023
Instructor: Dr. Lu Yang

Module 8 Securing User and Application Access (Sections 2-4)
Nov 3, 2023

Housekeeping items and feedback

1. Start recording
2. Start working on your term project
3. Remind me of your messages on Teams. But try to ask TAs questions first.

AWS Academy Cloud Architecting

Module 8: Securing User and Application Access

Module overview



Sections

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

Module objectives



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

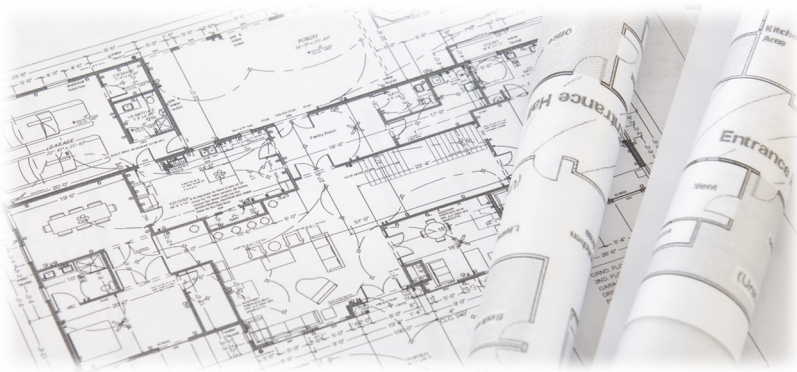
- Explain the purpose of AWS Identity and Access Management (IAM) users, groups, and roles
- Describe how to allow user federation within an architecture to increase security
- Recognize how AWS Organizations service control policies (SCPs) increase security within an architecture
- Describe how to manage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

Module 8: Securing User and Application Access

Section 1: Architectural need

Café business requirement

The café needs to define what level of access users and systems should have across cloud resources and then put these access controls into place across the AWS account.

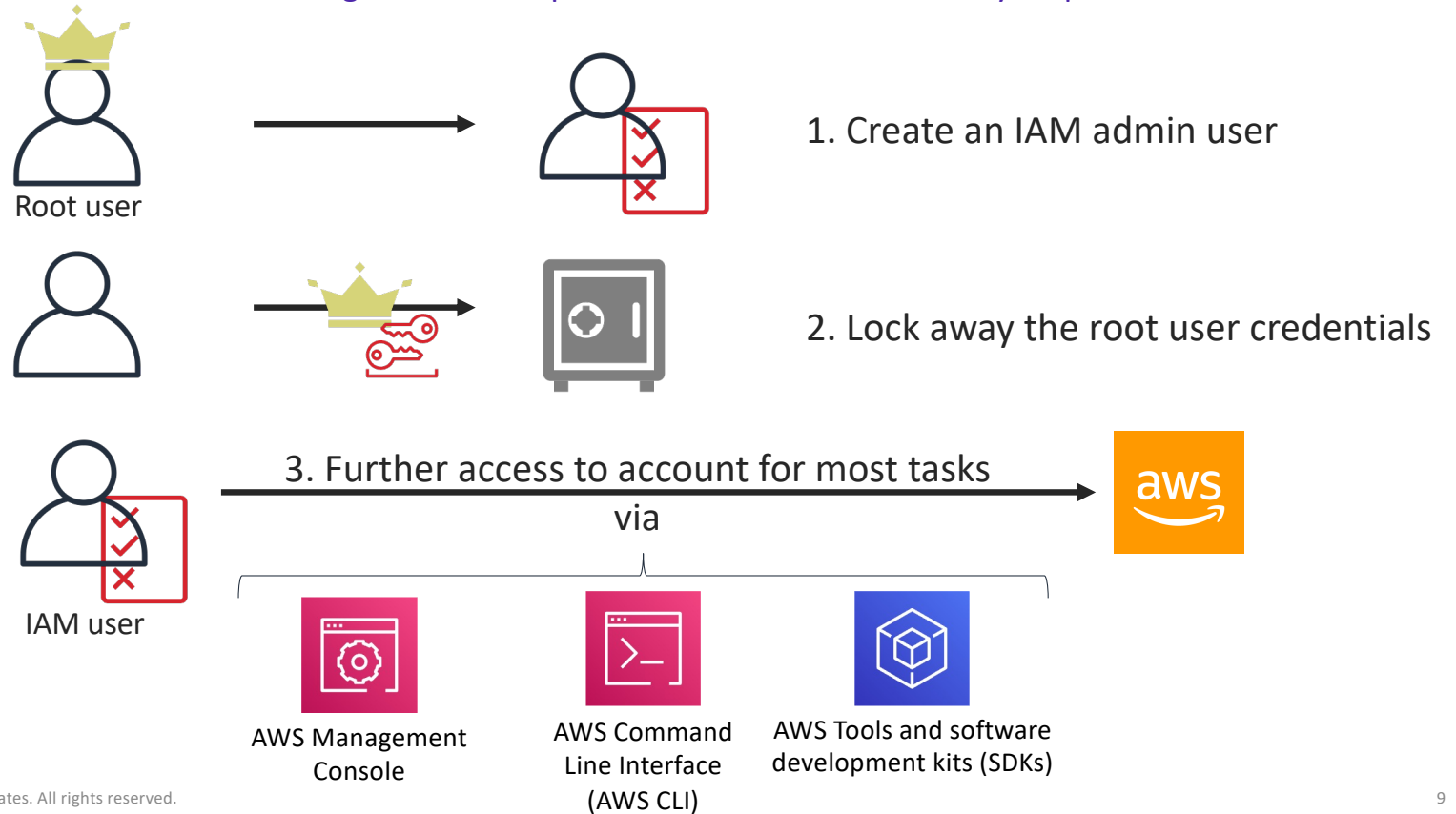


Module 8: Securing User and Application Access

Section 2: Account users and IAM

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.



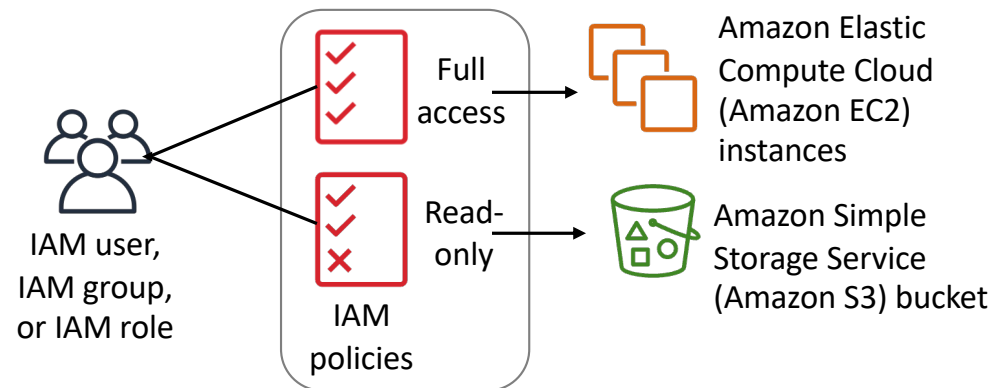
IAM policy

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 user, application, or service.



IAM permissions

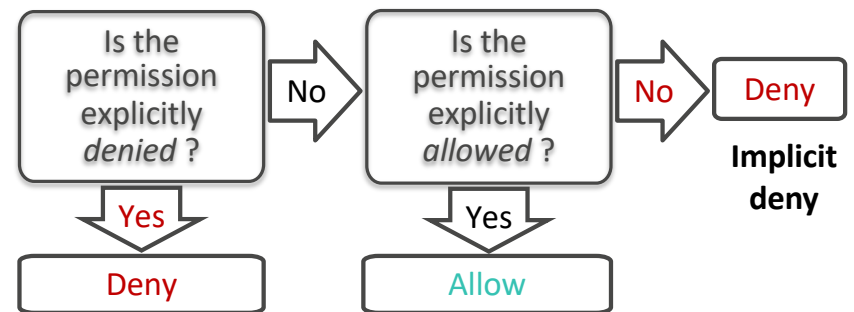


IAM policy

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 (IAM users, IAM groups, IAM roles, or AWS services)
 - **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, role, or service
- Types of policies
 - AWS managed
 - Customer managed
 - Inline



Resource-based policies

- Attached to AWS resources
 - Example: Attach to an Amazon S3 bucket
- No managed resource-based policies
- Always an inline policy

IAM policy document structure (1/2)

```
{
  "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"
 }
}`

IAM policy document structure (2/2)



- IAM JSON policy elements: Version
- IAM JSON policy elements: Id
- IAM JSON policy elements: Statement
- IAM JSON policy elements: Sid
- IAM JSON policy elements: Effect
- AWS JSON policy elements: Principal
- AWS JSON policy elements: NotPrincipal
- IAM JSON policy elements: Action
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Resource
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Condition
- Variables and tags
- Supported data type

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. Go to <https://docs.aws.amazon.com/IAM/latest/UserGuide/> and in the left navigation expand *Reference > Policy Reference > Actions, Resources, and Condition Keys*. Choose *Identity And Access Management*. Scroll to the Actions Defined by Identity And Access Management list.

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"
  }]
}
```

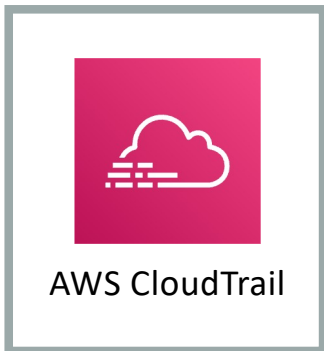
1. What actions does the policy allow?
 - **ANSWER: It does not allow you to do anything (the effect is to Deny).**
2. Say that the policy included an additional statement object, like this example:

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

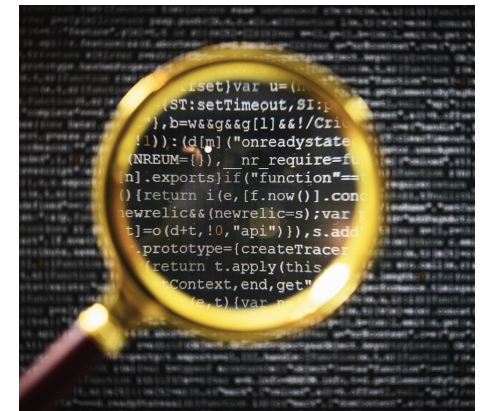
How would the policy restrict the access granted to you by this additional statement?

- **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.**

AWS CloudTrail



- Logs and monitors user activity
- Provides event history of AWS account
 - Actions taken through the AWS Management Console, SDKs, AWS CLI
 - Increases visibility into your user and resource activity
 - 90-day event history provided by default, at no cost
- Identify
 - The identity of the user (who deleted the instance)
 - The start time of the AWS API call (when the instance got deleted)
 - The source IP address
 - The request parameters (e.g., instance ID)
 - The response parameters returned by the service
- Helpful tool to
 - Perform security analysis
 - Discover which calls were blocked (for example, by IAM policies)



AWS CloudTrail and CloudWatch



Parameters	AWS CloudWatch	AWS CloudTrail
Definition	CloudWatch is a monitoring service for AWS resources and applications.	CloudTrail is a web service that records API activity in your AWS account.
Monitoring Tool	CloudWatch monitors applications and infrastructure performance in the AWS environment.	CloudTrail monitors actions in the AWS environment.
Usage	With CloudWatch, we can:Collect and track metricsCollect and monitor log filesSet alarms and visualize	CloudTrail answers these questions:Who made a request?Which services were used?What actions were performed?What were the parameters for those actions? What were the response elements returned by the AWS service?
Logs or Event delivery Rate	CloudWatch delivers metrics data in 5-minute or 1-minute periods for essential monitoring and detailed monitoring, respectively.By default, log data will be sent by the CloudWatch Logs agent every five seconds.	CloudTrail delivers an event within 15 minutes of the API call.
Example	Suppose you have a web app running in your AWS environment; AWS CloudWatch can monitor bandwidth utilization, performance, and the traffic parameters of your app.	If your EC2 instance's security is compromised by an attacker, you can identify the culprit with the help of historical CloudTrail data Logs.
Integration with Other Services	EC2 instances, autoscaling, load balancers, AWS SNS, SQS, AWS RDS, AWS S3, AWS DynamoDB, AWS CloudTrail, other AWS resources.	AWS CloudWatch, AWS ElasticSearch, AWS Lambda, third-party monitoring platforms, AWS SNS, AWS, SQS, etc.

Reference:
<https://www.opsramp.com/guides/aws-monitoring-tool/cloudtrail-vs-cloudwatch/>

Section 2 key takeaways



- Avoid using the **account root user** for common tasks. Instead, create and use IAM user credentials.
- **Permissions** for accessing AWS account resources are defined in one or more IAM policy documents.
 - Attach IAM policies to IAM users, groups, or roles.
- When IAM determines permissions, an explicit **Deny** will always override any **Allow** statement.
- It is a best practice to follow the **principle of least privilege** when you grant access.


Module 8: Securing User and Application Access

Section 3: Organizing users

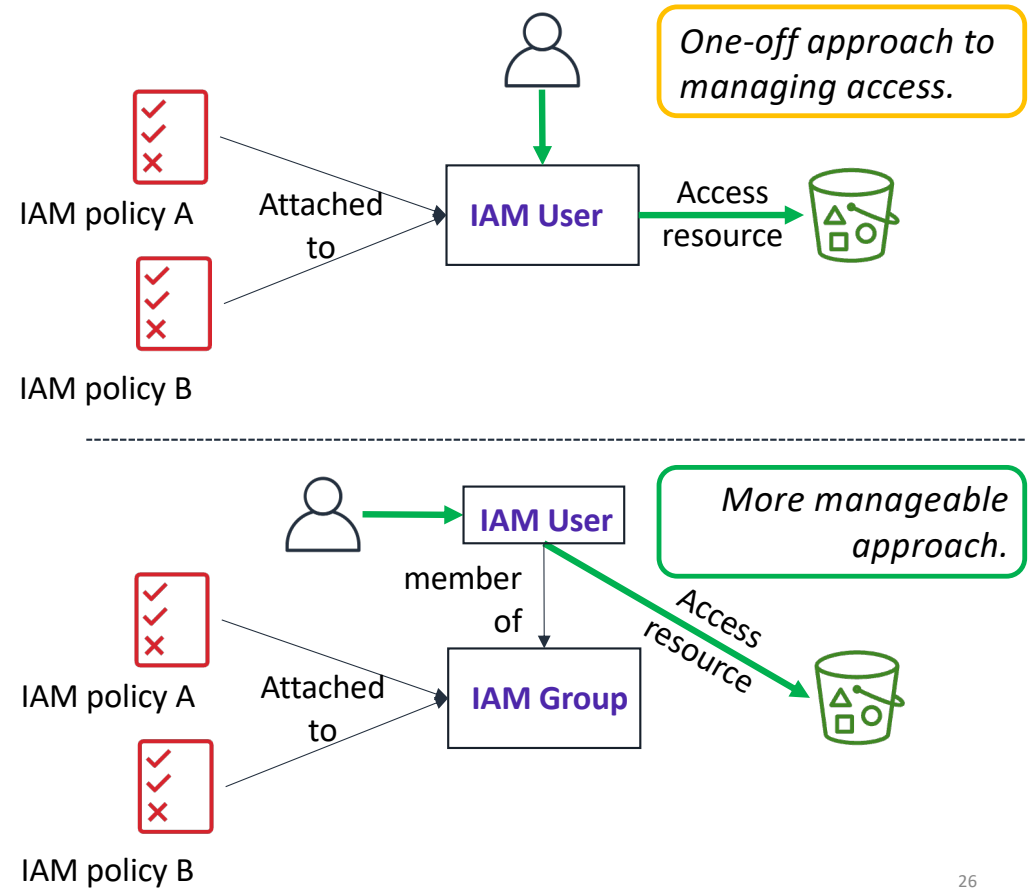
IAM groups

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

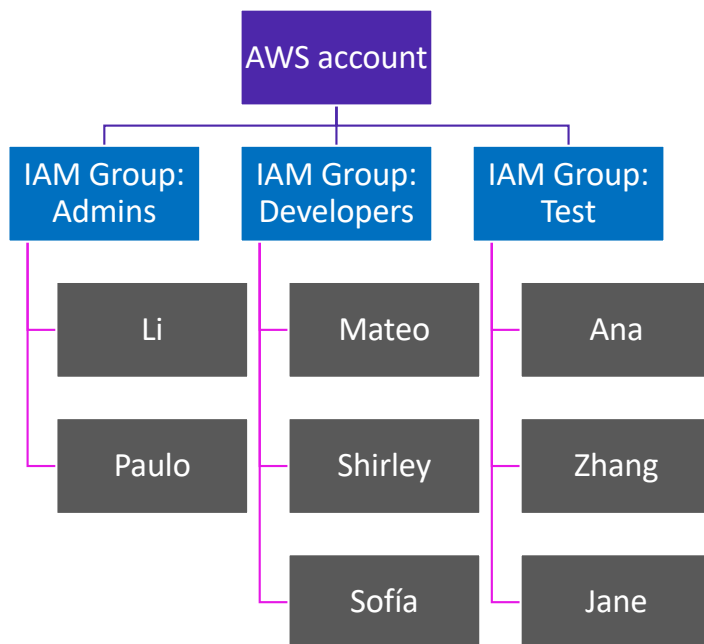
- Add the user to a group to apply standard access based on job function
- Optionally attach an additional policy to the user for needed exceptions



Example IAM groups

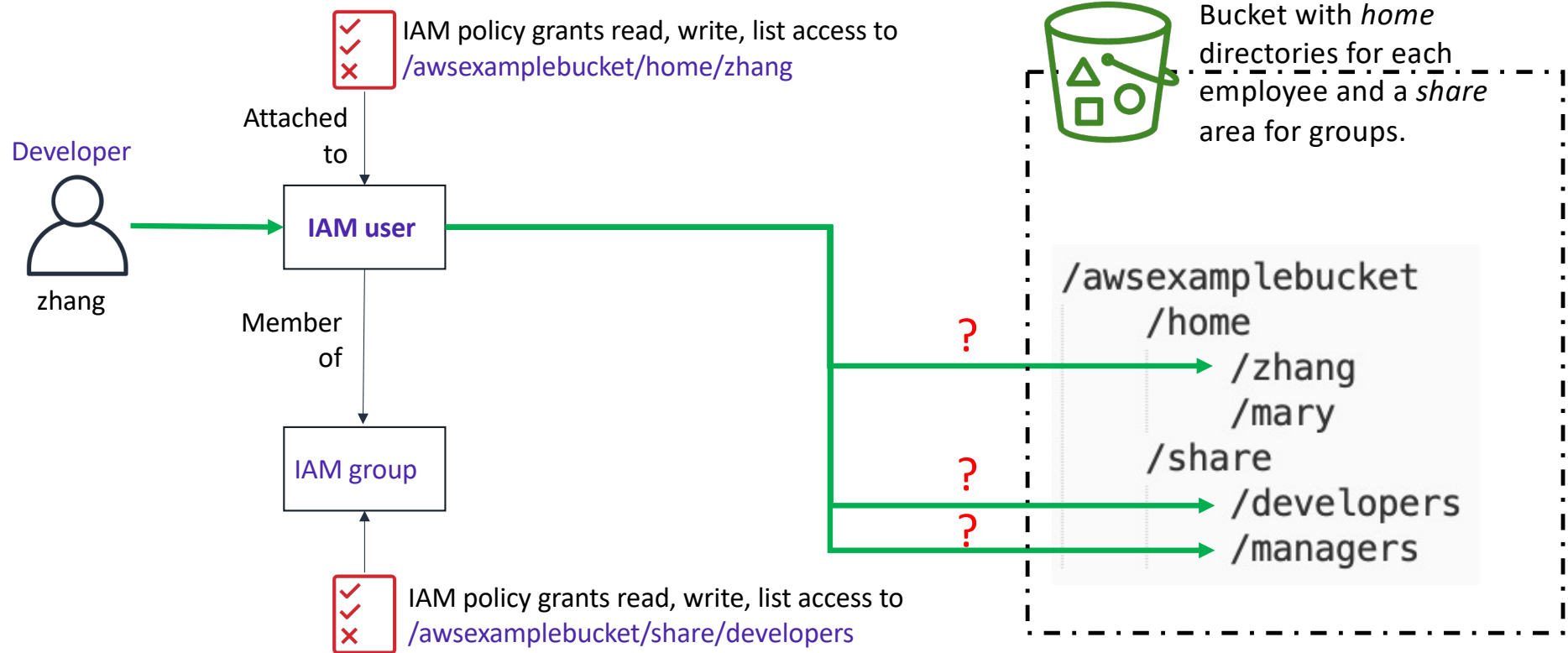


Tip: Create groups that reflect job functions



- 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

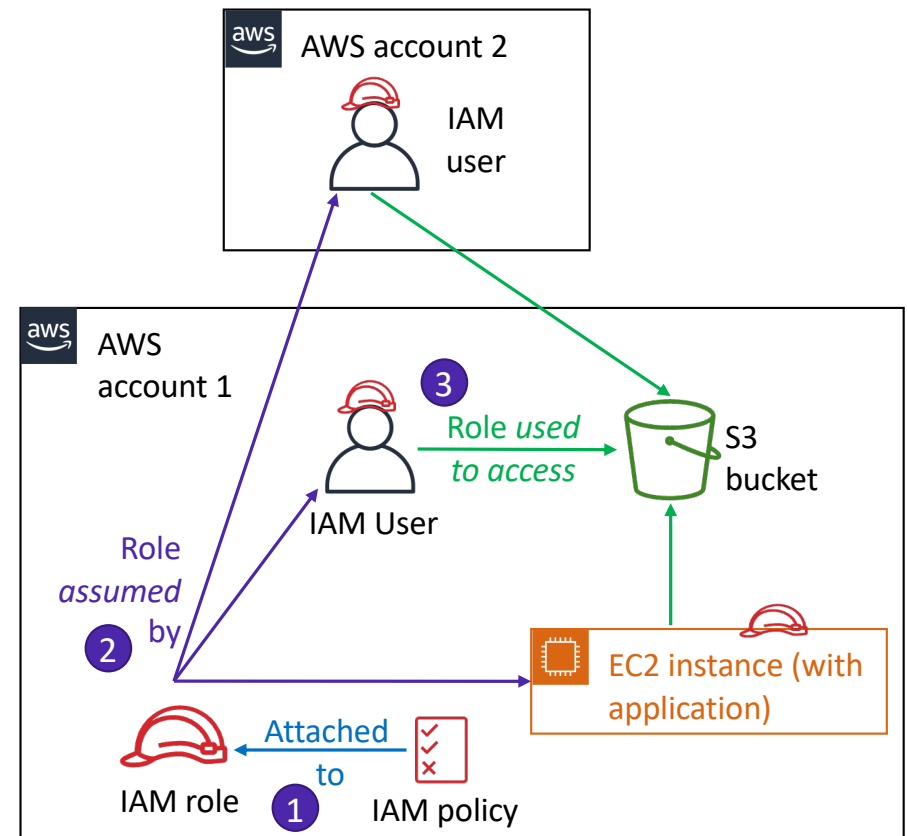


Module 8: Securing User and Application Access

Section 4: 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)



Demonstration: EC2 Instance Profile (how to attach an IAM role to an EC2 instance to access an S3 bucket)



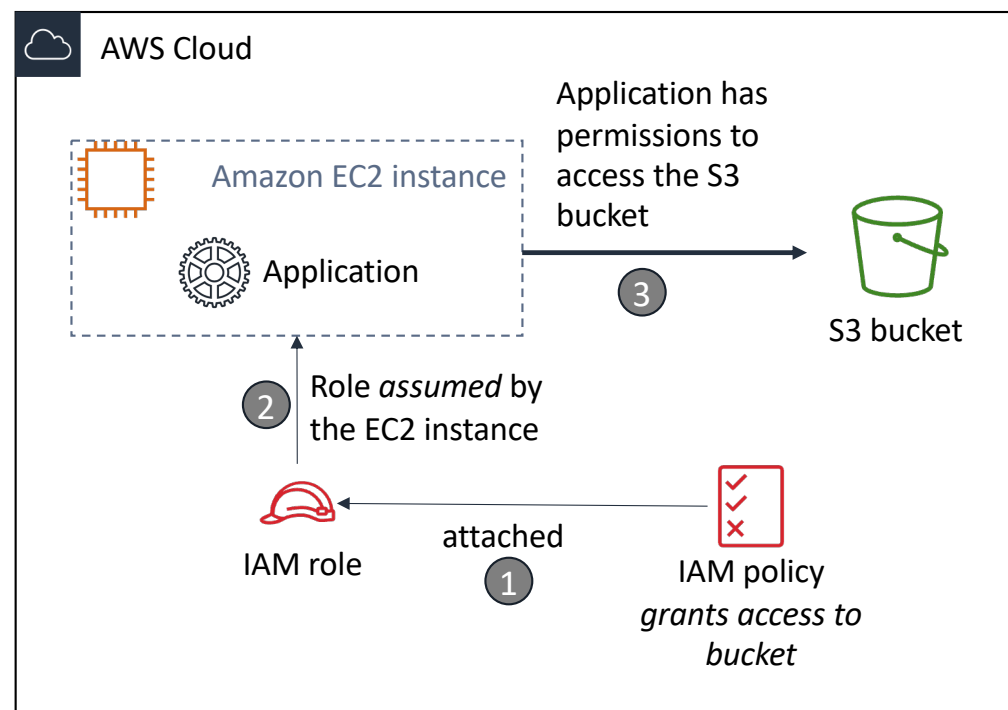
Summary: EC2 instance profile demonstration

Scenario:

- An application that runs on an EC2 instance needs access to an S3 bucket

Solution:

- Define an IAM policy that grants access to the S3 bucket
- Attach the policy to a role
- Allow the EC2 instance to assume the role



Grant permissions to assume a role

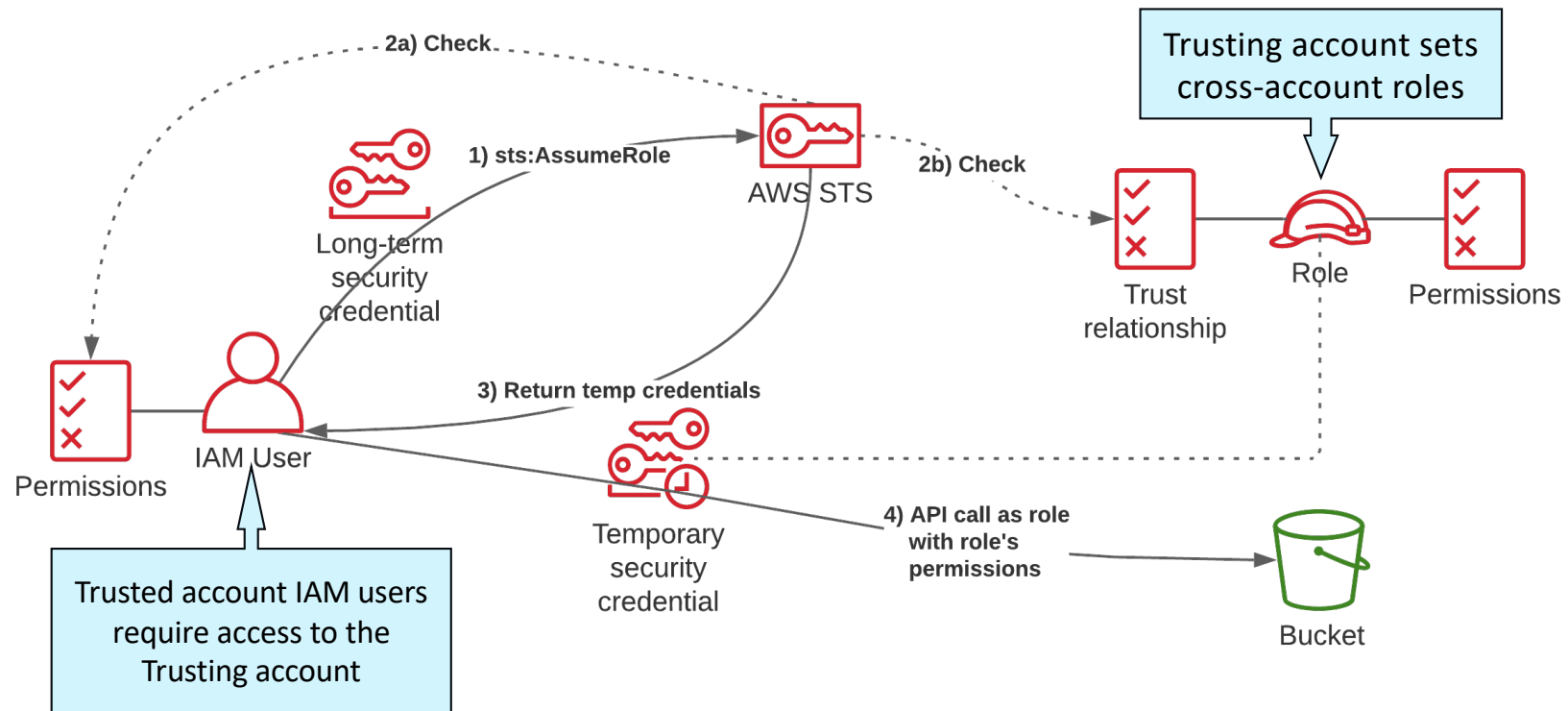


AWS Security
Token Service
(AWS STS)

- 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 (cross-account) or for users that you authenticate (federated users)
- AWS STS security tokens are typically used for identity federation, providing cross-account access and for resources related to EC2 instances that require access by other applications.
- 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*"  
  }  
}
```

How IAM roles are assumed? (Optional reading)

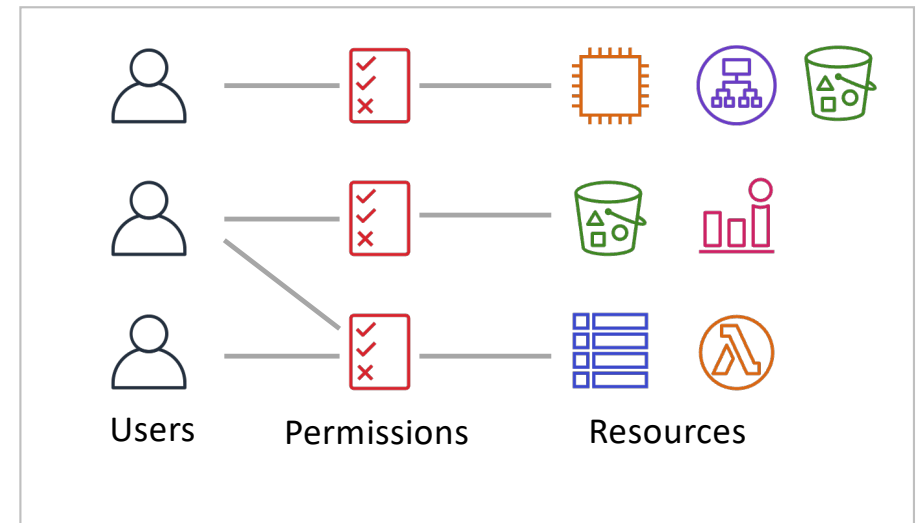


Reference: <https://www.tecracer.com/blog/2021/08/iam-what-happens-when-you-assume-a-role.html>

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)



Best practice: Tagging

- A tag consists of a name and (optionally) a value
 - Can be applied to **resources** across your AWS accounts
 - Tag keys and values are returned by many different API operations
- Define *custom* tags
- Multiple practical uses
 - Billing, filtered views, access control, etc.
- Example tags applied to an EC2 instance:
 - Name = web server
 - Project = unicorn
 - Stack = dev

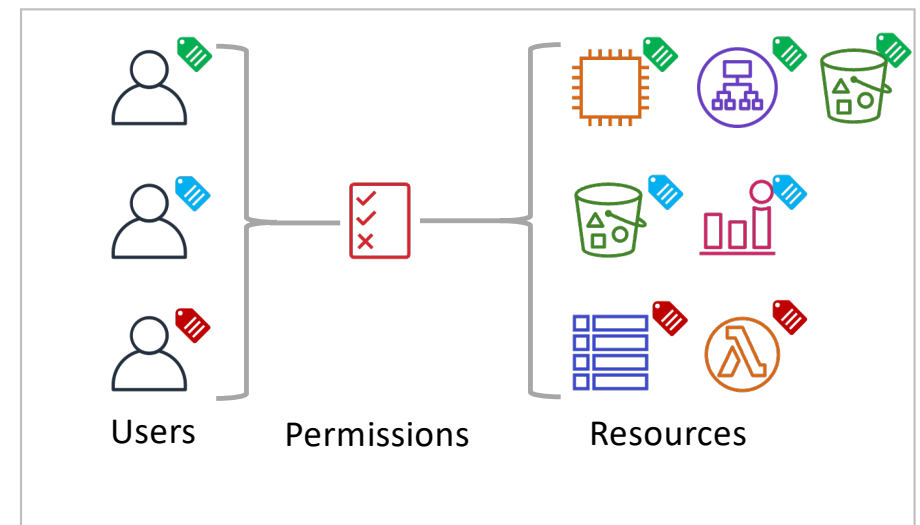
- Tags can also be applied to **IAM users** or **IAM roles**, for example –

The screenshot shows the 'Add user' console in AWS IAM. It is on step 3 of a 5-step process. The 'Add tags (optional)' section is active. It includes a description of IAM tags and a table for adding key-value pairs. Two tags are already added: 'CostCenter' with value '1234' and 'EmailID' with value 'john@example.com'. There is an 'Add new key' row at the bottom. Navigation buttons 'Cancel', 'Previous', and 'Next: Review' are at the bottom right.

Key	Value (optional)	Remove
CostCenter	1234	✕
EmailID	john@example.com	✕
Add new key		

Attribute-based access control (ABAC)

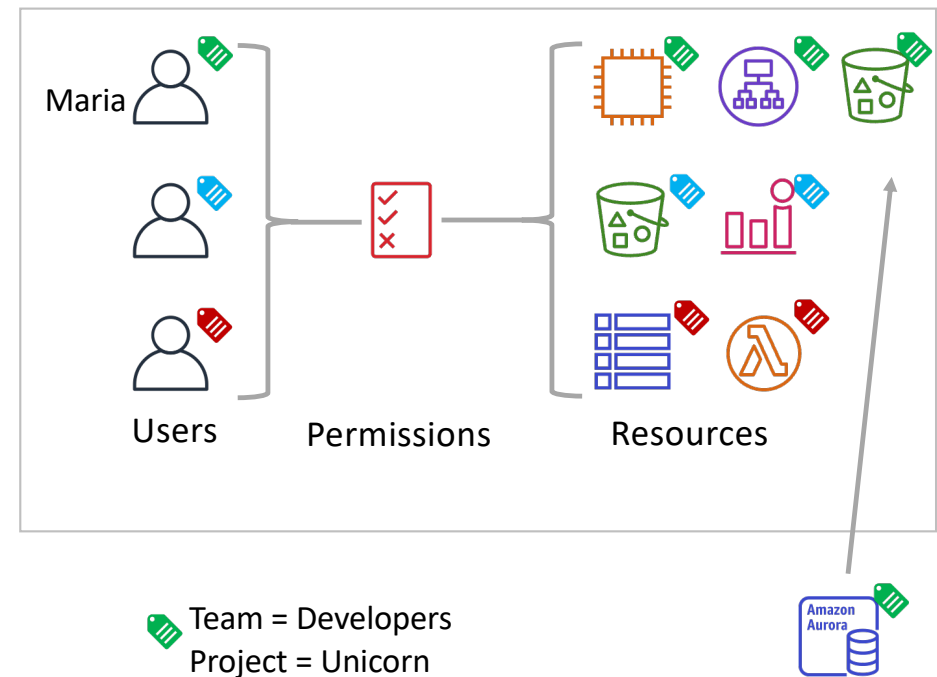
- Highly scalable approach to access control
 - Attributes are a key or a key-value pair, such as a tag
 - Example attributes –
 - Team = Developers
 - Project = Unicorn
- Permissions (policy) rules are easier to maintain with ABAC than with RBAC
- Benefits
 - Permissions automatically apply, based on 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



Demonstration: Implementing AWS IAM Attribute-Based Access Control:

https://www.youtube.com/watch?v=aO6j68USsfY&list=PLzde74P_a04cKnuXyi--fkloY1sxztyqL&index=2

