AWS IAM

Neal

IAM is the AWS Identity and Access Management Service.

IAM is used to securely control individual and group access to AWS resources.

IAM makes it easy to provide multiple users secure access to AWS resources.

IAM can be used to manage:

- Users.
- Groups.
- Access policies.
- Roles.
- User credentials.

- User password policies.
- Multi-factor authentication (MFA).
- API keys for programmatic access (CLI).

Provides centralized control of your AWS account.

Enables shared access to your AWS account.

By default new users are created with NO access to any AWS services – they can only login to the AWS console.

Permission must be explicitly granted to allow a user to access an AWS service.

IAM users are individuals who have been granted access to an AWS account.

Each IAM user has three main components:

- A username.
- A password.
- Permissions to access various resources.

You can apply granular permissions with IAM.

You can assign users individual security credentials such as access keys, passwords, and multi-factor authentication devices.

IAM is not used for application-level authentication.

Identity Federation (including AD, Facebook etc). can be configured allowing secure access to resources in an AWS account without creating an IAM user account.

Multi-factor authentication (MFA) can be enabled/enforced for the AWS account and for individual users under the account.

MFA uses an authentication device that continually generates random, six-digit, single-use authentication codes.

You can authenticate using an MFA device in the following three ways:

- Through the **AWS Management Console** – the user is prompted for a user name, password, and authentication code.
- Using the **AWS API** – restrictions are added to IAM policies and developers can request temporary security credentials and pass MFA parameters in their AWS STS API requests.
- Using the **AWS CLI** by obtaining temporary security credentials from STS (aws sts get-session-token).

Want to see how to setup MFA? In the brief AWS Hands-on Labs video tutorial below, you'll learn how to activate a virtual Multi-factor Authentication (MFA) for your AWS Root Account. In under 5 minutes, we cover: Deleting the Root Account Access Key and Activating Multi-Factor Authentication.

It is a best practice to use MFA for all users and to use U2F or hardware MFA devices for all privileged users.

IAM is universal (global) and does not apply to regions.

IAM is eventually consistent.

IAM replicates data across multiple data centers around the world.

The "root account" is the account created when you setup the AWS account. It has complete Admin access and is the only account that has this access by default.

It is a best practice to not use the root account for anything other than billing.

Power user access allows all permissions except the management of groups and users in IAM.

Temporary security credentials consist of the AWS access key ID, secret access key, and security token.

IAM can assign temporary security credentials to provide users with temporary access to services/resources.

To sign-in you must provide your account ID or account alias in addition to a user name and password.

The sign-in URL includes the account ID or account alias, e.g.:

https://My_AWS_Account_ID.signin.aws.amazon.com/console/.

Alternatively you can sign-in at the following URL and enter your account ID or alias manually:

https://console.aws.amazon.com/.

IAM integrates with many different AWS services.

IAM supports PCI DSS compliance.

AWS recommend that you use the AWS SDKs to make programmatic API calls to IAM.

However, you can also use the IAM Query API to make direct calls to the IAM web service.

IAM Elements

Principals:

- An entity that can take an action on an AWS resource.
- Your administrative IAM user is your first principal.
- You can allow users and services to assume a role.
- IAM supports federated users.
- IAM supports programmatic access to allow an application to access your AWS account.
- IAM users, roles, federated users, and applications are all AWS principals.

Requests:

- Principals send requests via the Console, CLI, SDKs, or APIs.
- Requests are:
  - Actions (or operations) that the principal wants to perform.
  - Resources upon which the actions are performed.
  - Principal information including the environment from which the request was made.
- Request context – AWS gathers the request information:
  - Principal (requester).
  - Aggregate permissions associated with the principal.
  - Environment data, such as IP address, user agent, SSL status etc.
  - Resource data, or data that is related to the resource being requested.

Authentication:

- A principal sending a request must be authenticated to send a request to AWS.
- To authenticate from the console, you must sign in with your user name and password.
- To authenticate from the API or CLI, you must provide your access key and secret key.

Authorization:

- IAM uses values from the request context to check for matching policies and determines whether to allow or deny the request.
- IAM policies are stored in IAM as JSON documents and specify the permissions that are allowed or denied.
- IAM policies can be:
  - User (identity) based policies.
  - Resource-based policies.
- IAM checks each policy that matches the context of your request.
- If a single policy has a deny action IAM denies the request and stops evaluating (explicit deny).
- Evaluation logic:

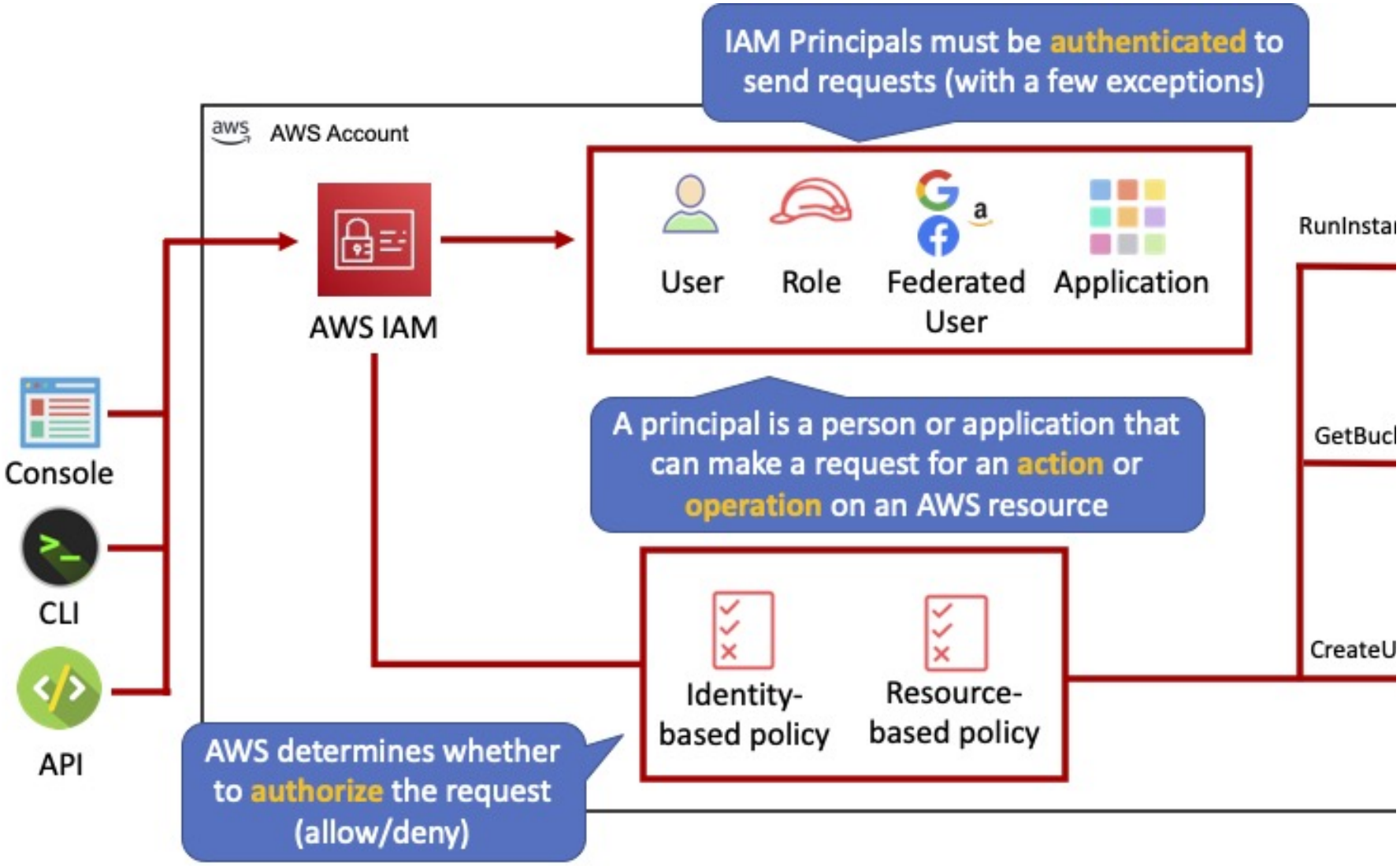
    - By default all requests are denied (implicit deny).
    - An explicit allow overrides the implicit deny.
    - An explicit deny overrides any explicit allows.
- Only the root user has access to all resources in the account by default.

Actions:

- Actions are defined by a service.
- Actions are the things you can do to a resource such as viewing, creating, editing, deleting.
- Any actions on resources that are not explicitly allowed are denied.
- To allow a principal to perform an action you must include the necessary actions in a policy that applies to the principal or the affected resource.

Resources:

- A resource is an entity that exists within a service.
- E.g. EC2 instances, S3 buckets, IAM users, and DynamoDB tables.
- Each AWS service defines a set of actions that can be performed on the resource.
- After AWS approves the actions in your request, those actions can be performed on the related resources within your account.



Authentication Methods

Console password:

- A password that the user can enter to sign into interactive sessions such as the AWS Management

Console.
- You can allow users to change their own passwords.
- You can allow selected IAM users to change their passwords by disabling the option for all users and using an IAM policy to grant permissions for the selected users.

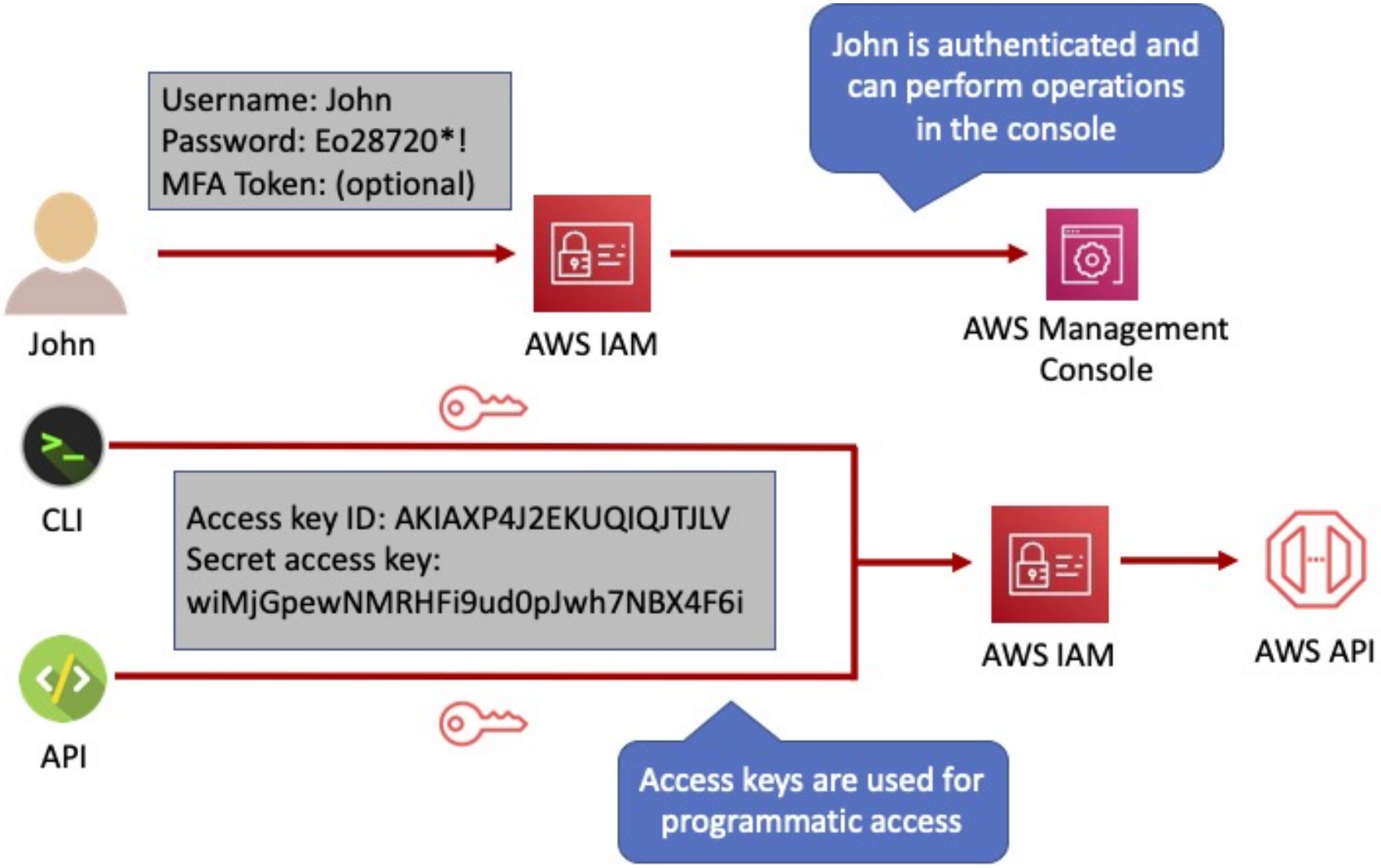
Access Keys:

- A combination of an **access key ID** and a **secret access key.**
- You can assign two active access keys to a user at a time.
- These can be used to make programmatic calls to AWS when using the **API** in program code or at a command prompt when using the **AWS CLI** or the **AWS PowerShell** tools.
- You can create, modify, view, or rotate access keys.
- When created IAM returns the access key ID and secret access key.
- The secret access is returned only at creation time and if lost a new key must be created.
- Ensure access keys and secret access keys are stored securely.
- Users can be given access to change their own keys through IAM policy (not from the console).
- You can disable a user's access key which prevents it from being used for API calls.

Server certificates:

- SSL/TLS certificates that you can use to authenticate with some AWS services.
- AWS recommends that you use the AWS Certificate Manager (ACM) to provision, manage and deploy your server certificates.
- Use IAM only when you must support HTTPS connections in a region that is not supported by ACM.

The following diagram shows the different methods of authentication available with IAM:



IAM Users

An IAM user is an entity that represents a person or service.

Can be assigned:

- An access key ID and secret access key for programmatic access to the AWS API, CLI, SDK, and other development tools.
- A password for access to the management console.

By default users cannot access anything in your account.

The account root user credentials are the email address used to create the account and a password.

The root account has full administrative permissions, and these cannot be restricted.

Best practice for root accounts:

- Don’t use the root user credentials.
- Don’t share the root user credentials.
- Create an IAM user and assign administrative permissions as required.
- Enable MFA.

IAM users can be created to represent applications, and these are known as “service accounts”.

You can have up to 5000 users per AWS account.

Each user account has a friendly name and an ARN which uniquely identifies the user across AWS.

A unique ID is also created which is returned only when you create the user using the API, Tools for Windows PowerShell, or the AWS CLI.

You should create individual IAM accounts for users (best practice not to share accounts).

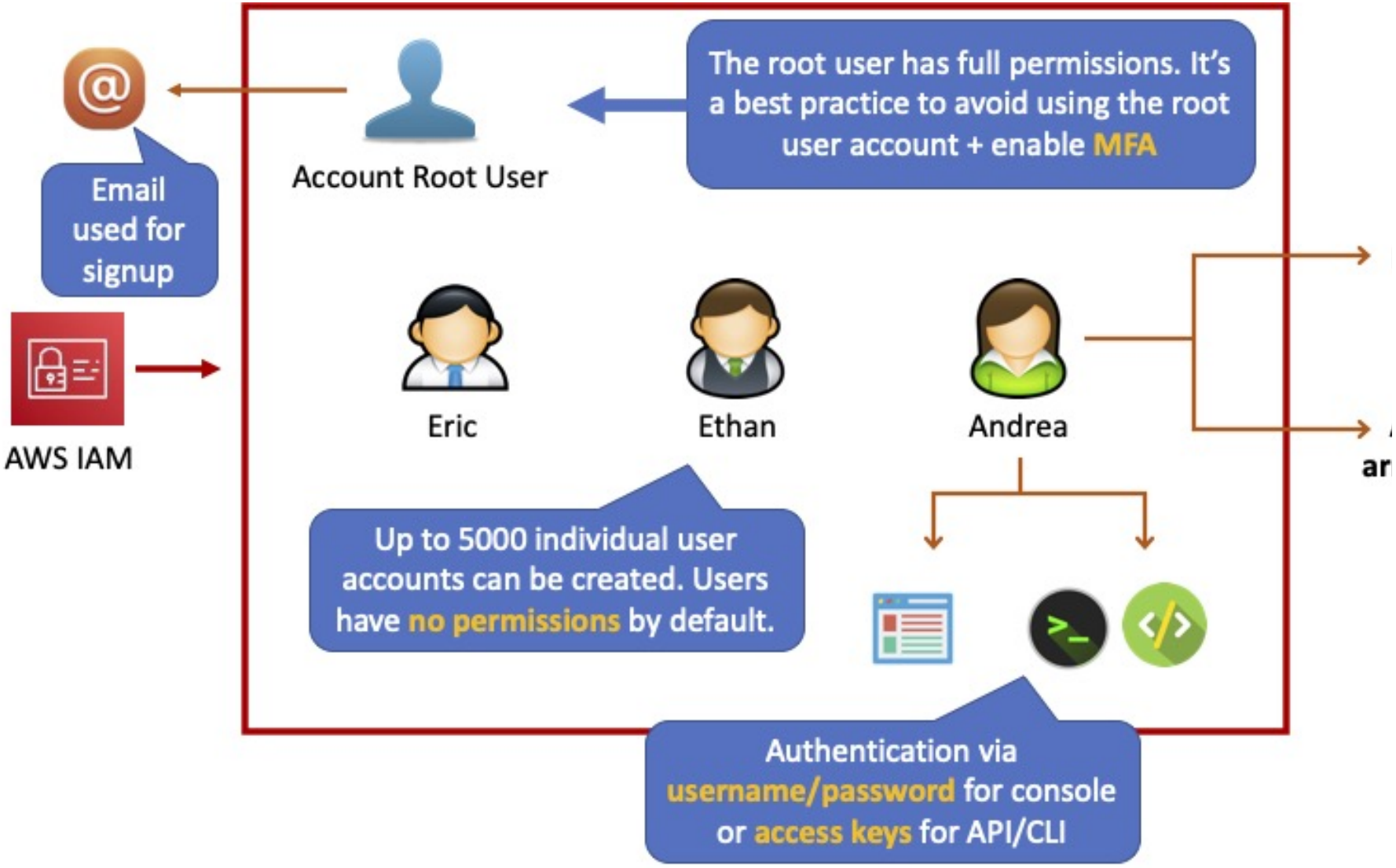
The Access Key ID and Secret Access Key are not the same as a password and cannot be used to login to the AWS console.

The Access Key ID and Secret Access Key can only be generated once and must be regenerated if lost.

A password policy can be defined for enforcing password length, complexity etc. (applies to all users).

You can allow or disallow the ability to change passwords using an IAM policy.

Access keys and passwords should be changed regularly.



Groups

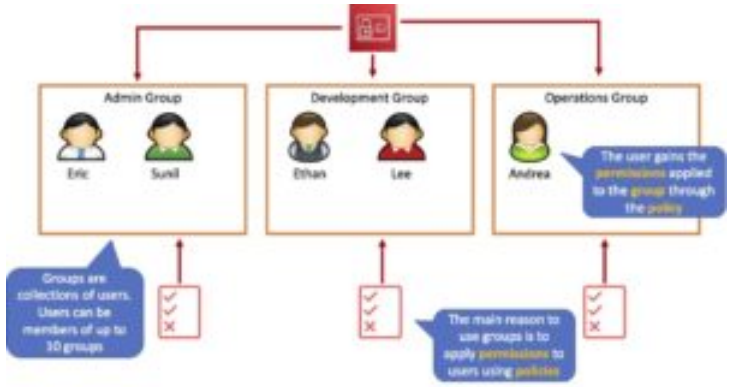
Groups are collections of users and have policies attached to them.

A group is not an identity and cannot be identified as a principal in an IAM policy.

Use groups to assign permissions to users.

Use the principal of least privilege when assigning permissions.

You cannot nest groups (groups within groups).



Roles

Roles are created and then “assumed” by trusted entities and define a set of permissions for making AWS service requests.

With IAM Roles you can delegate permissions to resources for users and services without using permanent credentials (e.g. user name and password).

IAM users or AWS services can assume a role to obtain temporary security credentials that can be used

to make AWS API calls.

You can delegate using roles.

There are no credentials associated with a role (password or access keys).

IAM users can temporarily assume a role to take on permissions for a specific task.

A role can be assigned to a federated user who signs in using an external identity provider.

Temporary credentials are primarily used with IAM roles and automatically expire.

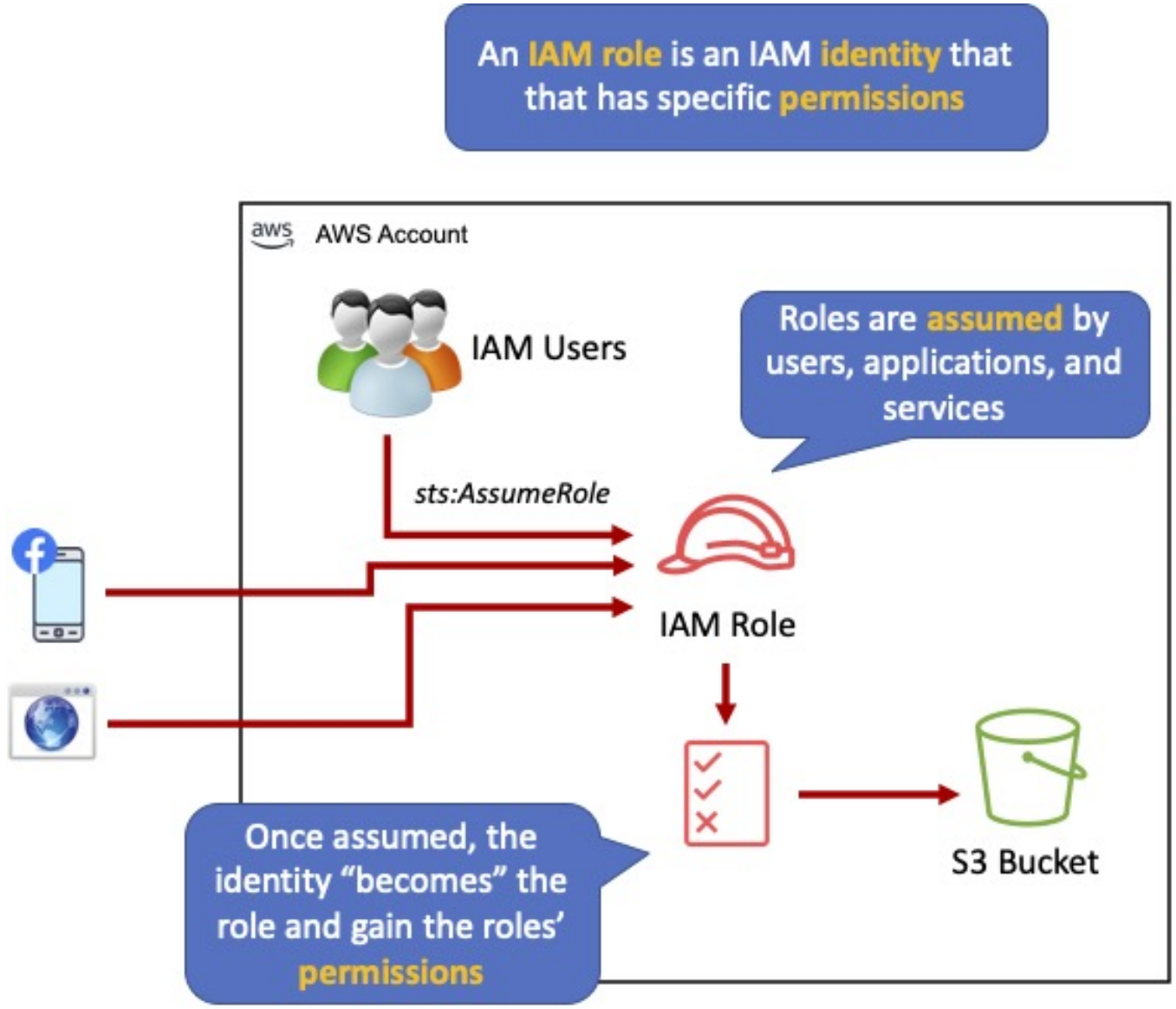
Roles can be assumed temporarily through the console or programmatically with the **AWS CLI**, **Tools for Windows PowerShell,** or **API.**

IAM roles with EC2 instances:

- IAM roles can be used for granting applications running on EC2 instances permissions to AWS API requests using instance profiles.
- Only one role can be assigned to an EC2 instance at a time.
- A role can be assigned at the **EC2 instance creation time or at any time afterwards.**
- When using the AWS CLI or API instance profiles must be created manually (it's automatic and transparent through the console).
- Applications retrieve temporary security credentials from the instance metadata.

Role Delegation:

- Create an IAM role with two policies:
  - Permissions policy – grants the user of the role the required permissions on a resource.
  - Trust policy – specifies the trusted accounts that are allowed to assume the role.
- Wildcards (*) cannot be specified as a principal.
- A permissions policy must also be attached to the user in the trusted account.



Policies

Policies are documents that define permissions and can be applied to users, groups, and roles.

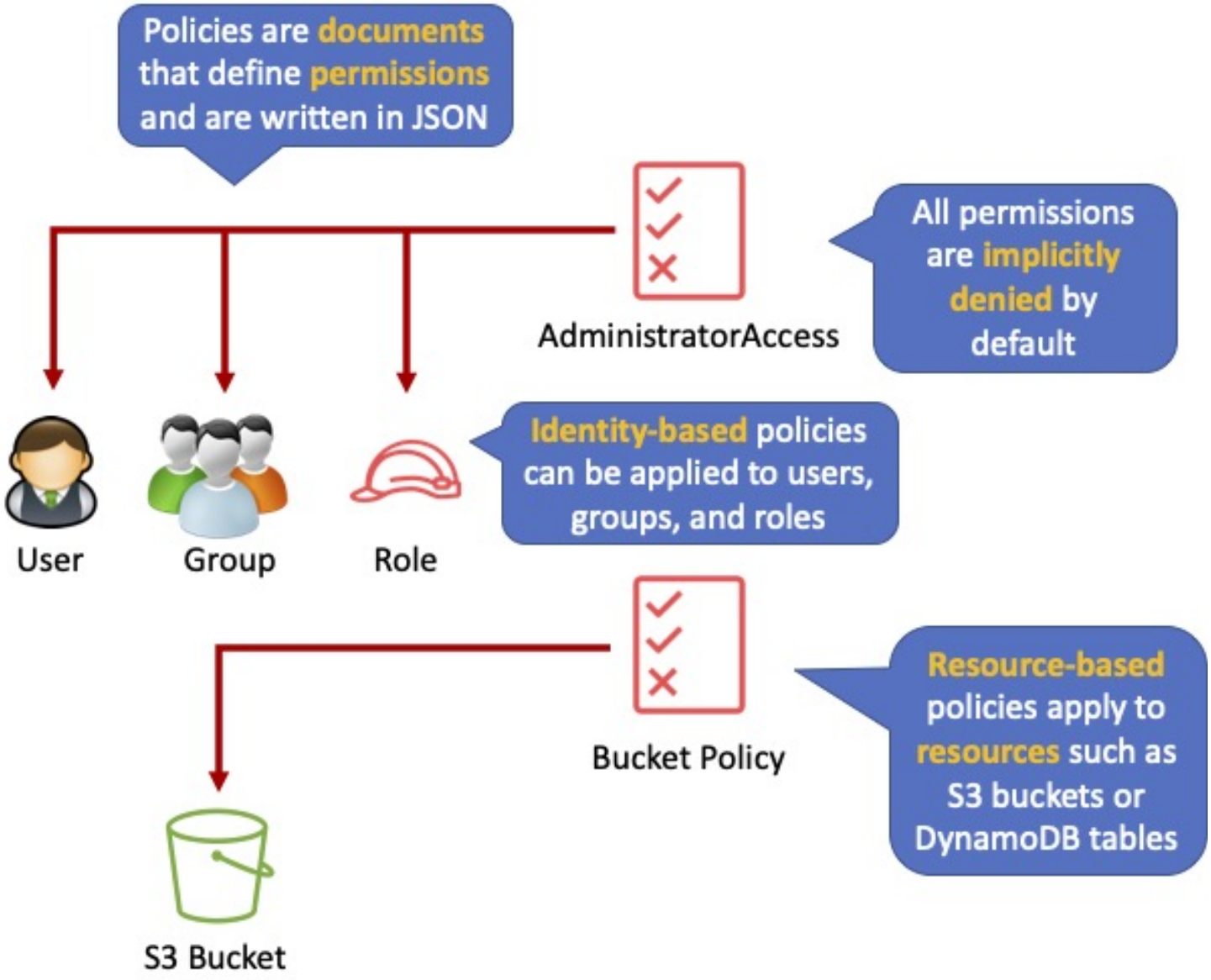
Policy documents are written in JSON (key value pair that consists of an attribute and a value).

All permissions are implicitly denied by default.

The most restrictive policy is applied.

The IAM policy simulator is a tool to help you understand, test, and validate the effects of access control policies.

The Condition element can be used to apply further conditional logic.



Inline Policies vs Managed Policies

There are 3 types of policies:

- Managed policies.
- Customer managed policies.
- Inline policies.

Managed Policy:

- Created and administered by AWS.
- Used for common use cases based on job function.
- Save you having to create policies yourself.
- Can be attached to multiple users, groups, or roles within and across AWS accounts.
- Cannot change the permissions assigned.

Customer Managed Policy:

- Standalone policy that you create and administer in your own AWS account.
- Can be attached to multiple users, groups, and roles – but only within your own account.
- Can be created by copying an existing managed policy and then customizing it.
- Recommended for use cases where the existing AWS Managed Policies don't meet the needs of your environment.

Inline Policy:

- Inline policies are embedded within the user, group, or role to which it is applied.
- Strict 1:1 relationship between the entity and the policy.
- When you delete the user, group, or role in which the inline policy is embedded, the policy will also be deleted.
- In most cases, AWS recommends using Managed Policies instead of inline policies.
- Inline policies are useful when you want to be sure that the permissions in a policy are not inadvertently assigned to any other user, group, or role.

AWS Managed and Customer Managed Policies

An AWS managed policy is a standalone policy that is created and administered by AWS.

Standalone policy means that the policy has its own Amazon Resource Name (ARN) that includes the policy name.

AWS managed policies are designed to provide permissions for many common use cases.

You cannot change the permissions defined in AWS managed policies.

Some AWS managed policies are designed for specific job functions.

The job-specific AWS managed policies include:

- Administrator.
- Billing.
- Database Administrator.
- Data Scientist.
- Developer Power User.
- Network Administrator.
- Security Auditor.
- Support User.
- System Administrator.
- View-Only User.

You can create standalone policies that you administer in your own AWS account, which we refer to as customer managed policies.

You can then attach the policies to multiple principal entities in your AWS account.

When you attach a policy to a principal entity, you give the entity the permissions that are defined in

the policy.

IAM Policy Evaluation Logic

By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access).

An explicit allow in an identity-based or resource-based policy overrides this default.

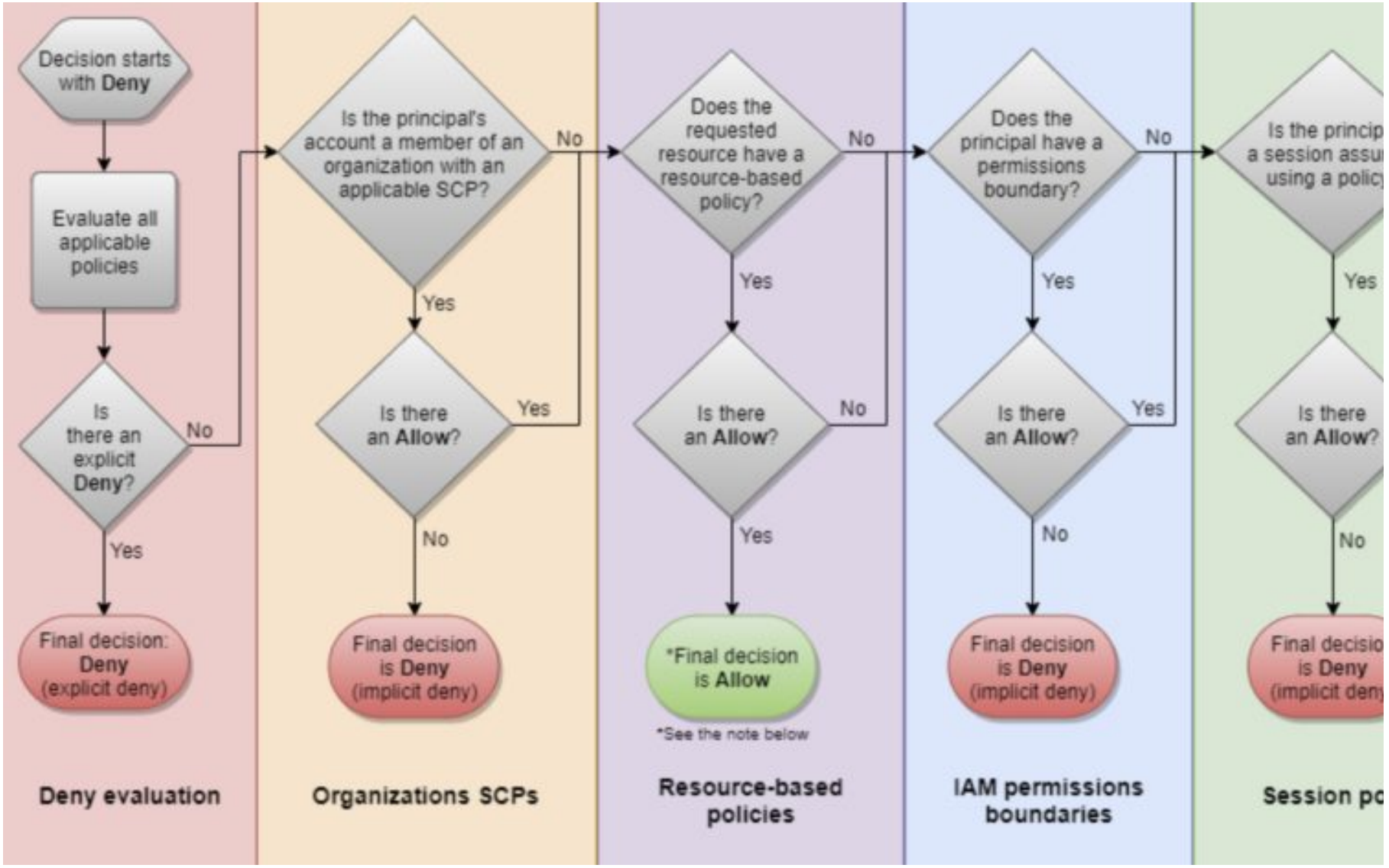
If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.

An explicit deny in any policy overrides any allows.

A few concepts should be known to understand the logic:

- **Identity-based policies** – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles).
- **Resource-based policies** – Resource-based policies grant permissions to the principal (account, user, role, or federated user) specified as the principal.
- **IAM permissions boundaries** – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role).
- **AWS Organizations service control policies (SCPs)** – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). Session policies – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user.

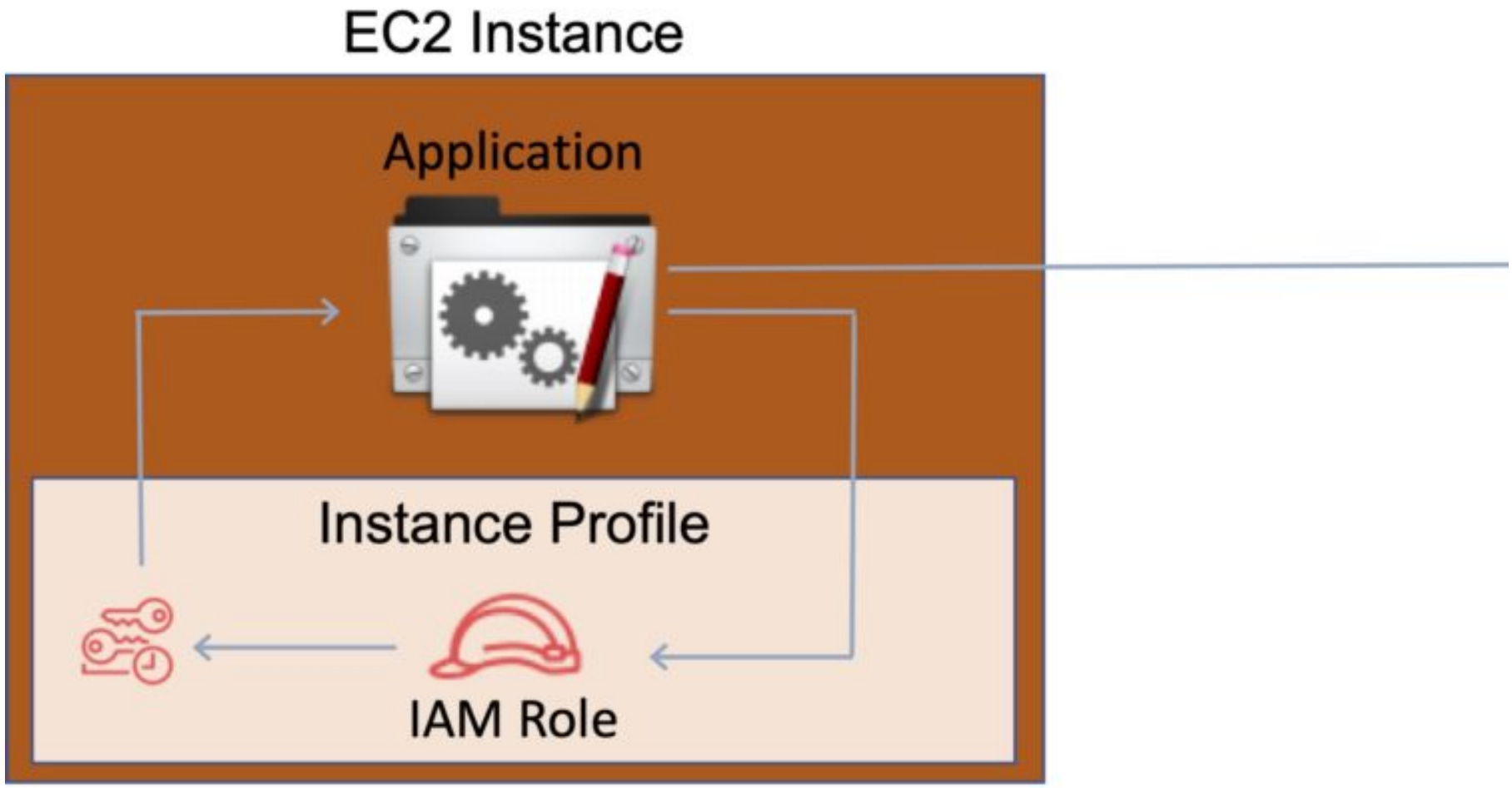
The following flowchart details the IAM policy evaluation logic:



IAM Instance Profiles

An instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.

An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles.



You can use the following AWS CLI commands to work with instance profiles in an AWS account:

- Create an instance profile: aws iam create-instance-profile
- Add a role to an instance profile: aws iam add-role-to-instance-profile
- List instance profiles: aws iam list-instance-profiles, aws iam list-instance-profiles-for-role

- Get information about an instance profile: aws iam get-instance-profile
- Remove a role from an instance profile: aws iam remove-role-from-instance-profile
- Delete an instance profile: aws iam delete-instance-profile

AWS Security Token Service

The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for IAM users or for users that you authenticate (federated users).

By default, AWS STS is available as a global service, and all AWS STS requests go to a single endpoint at https://sts.amazonaws.com

You can optionally send your AWS STS requests to endpoints in any region (can reduce latency).

Credentials will always work globally.

STS supports AWS CloudTrail, which records AWS calls for your AWS account and delivers log files to an S3 bucket.

Temporary security credentials work almost identically to long-term access key credentials that IAM users can use, with the following differences:

- Temporary security credentials are short-term.
- They can be configured to last anywhere from a few minutes to several hours.
- After the credentials expire, AWS no longer recognizes them or allows any kind of access to API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested.
- When (or even before) the temporary security credentials expire, the user can request new credentials, if the user requesting them still has permission to do so.

Advantages of STS are:

- You do not have to distribute or embed long-term AWS security credentials with an application.
- You can provide access to your AWS resources to users without having to define an AWS identity for them (temporary security credentials are the basis for IAM Roles and ID Federation).
- The temporary security credentials have a limited lifetime, so you do not have to rotate them or explicitly revoke them when they're no longer needed.
- After temporary security credentials expire, they cannot be reused (you can specify how long the credentials are valid for, up to a maximum limit).

The AWS STS API action returns temporary security credentials that consist of:

- An access key which consists of an access key ID and a secret ID.
- A session token.
- Expiration or duration of validity.
- Users (or an application that the user runs) can use these credentials to access your resources.

With STS you can request a session token using one of the following APIs:

- AssumeRole – can only be used by IAM users (can be used for MFA).
- AssumeRoleWithSAML – can be used by any user who passes a SAML authentication response that indicates authentication from a known (trusted) identity provider.
- AssumeRoleWithWebIdentity – can be used by an user who passes a web identity token that indicates authentication from a known (trusted) identity provider.
- GetSessionToken – can be used by an IAM user or AWS account root user (can be used for MFA).
- GetFederationToken – can be used by an IAM user or AWS account root user.

AWS recommends using Cognito for identity federation with Internet identity providers.

Users can come from three sources.

Federation (typically AD):

- Uses SAML 2.0.
- Grants temporary access based on the users AD credentials.
- Does not need to be a user in IAM.
- Single sign-on allows users to login to the AWS console without assigning IAM credentials.

Federation with Mobile Apps:

- Use Facebook/Amazon/Google or other OpenID providers to login.

Cross Account Access:

- Lets users from one AWS account access resources in another.
- To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.
- Or you must assume a role (identity-based policy) within that account with the permissions you need.

There are a couple of ways STS can be used.

Scenario 1:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker always authenticates with LDAP first, then with AWS STS.
3. Application then gets temporary access to AWS resources.

Scenario 2:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker authenticates with LDAP first, then gets an IAM role associated with the user.
3. Application then authenticates with STS and assumes that IAM role.
4. Application uses that IAM role to interact with the service.

Cross Account Access

Useful for situations where an AWS customer has separate AWS account – for example for development and production resources.

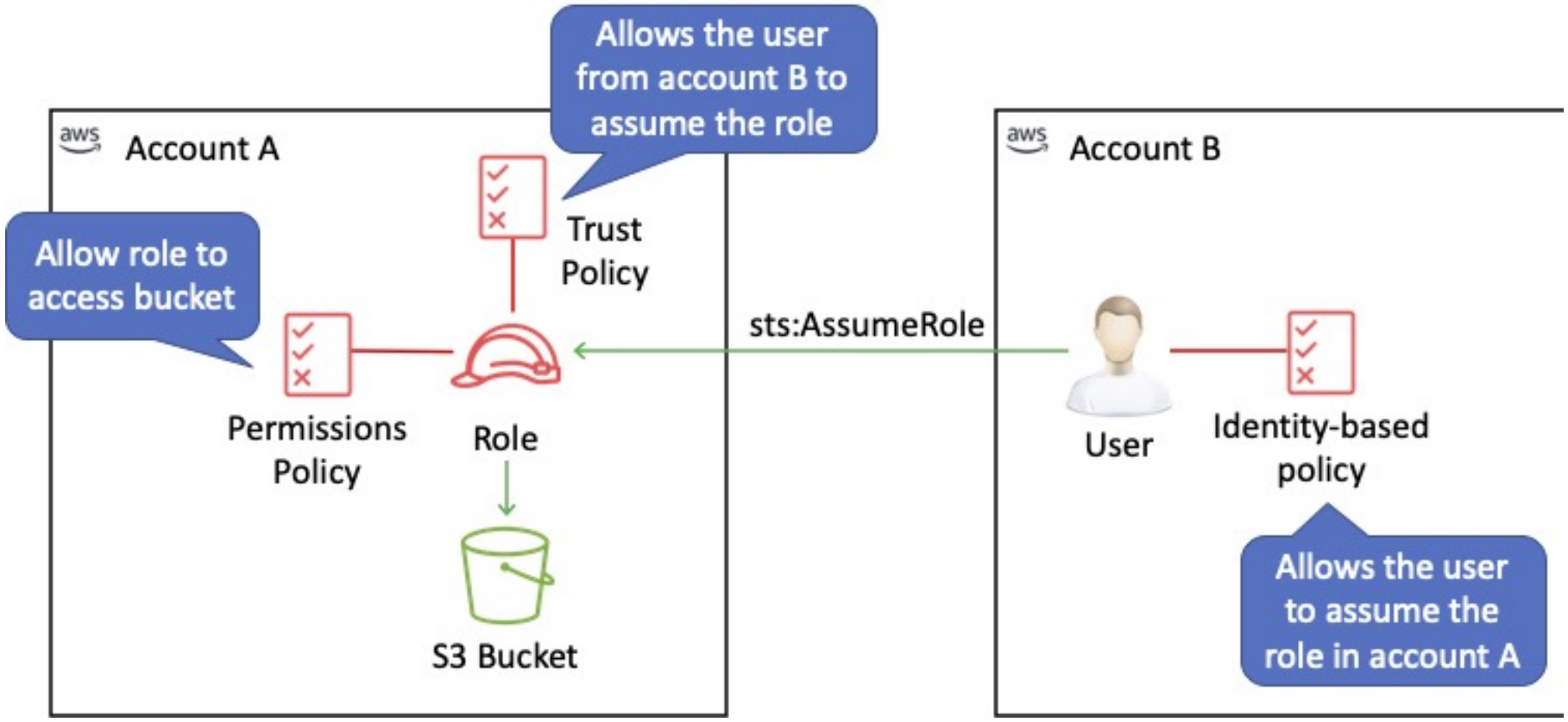
Cross Account Access makes is easier to work productively within a multi-account (or multi-role) AWS environment by making is easy to switch roles within the AWS Management Console.

Can sign-in to the console using your IAM user name and then switch the console to manage another account without having to enter another user name and password.

Lets users from one AWS account access resources in another.

To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.

Or you must assume a role (identity-based policy) within that account with the permissions you need.



IAM Best Practices

To secure AWS resources it is recommended that you follow these best practices:

- Lock away your AWS account root user access keys.
- Use roles to delegate permissions.
- Grant least privilege.
- Get started using permissions with AWS managed policies.
- Validate your policies.
- Use customer managed policies instead of inline policies.

- Use access levels to review IAM permissions.
- Configure a strong password policy for your users.
- Enable MFA.
- Use roles for applications that run on Amazon EC2 instances.
- Do not share access keys.
- Rotate credentials regularly.
- Remove unnecessary credentials.
- Use policy conditions for extra security.
- Monitor activity in your AWS account.

Related posts: