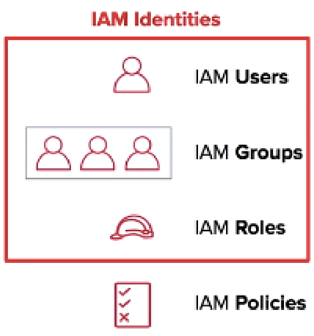
Identity Access Management (IAM)

IAM Simplified:

IAM offers a centralized hub of control within AWS and integrates with all other AWS Services. IAM comes with the ability to share access at various levels of permission and it supports the ability to use identity federation (the process of delegating authentication to a trusted external party like Facebook or Google) for temporary or limited access. IAM comes with MFA support and allows you to set up custom password rotation policy across your entire organization. It is also PCI DSS compliant i.e. payment card industry data security standard. (passes government mandated credit card security regulations).

IAM Entities:

- When accessing AWS, the root account should **never** be used. Users must be created with the proper permissions. IAM is central to AWS
- **Users**: A physical person
- **Groups**: Functions (admin, devops) Teams (engineering, design) which contain a group of users
- **Roles**: Internal usage within AWS resources
  - **Cross Account Roles**: roles used to assumed by another AWS account in order to have access to some resources in our account
- **Policies (JSON documents)**: Defines what each of the above can and cannot do. **Note**: IAM has predefined managed policies
  - There are 3 types of policies:
    - AWS Managed
    - Customer Managed
    - Inline Policies
- **Resource Based Policies**: policies attached to AWS services such as S3, SQS
- **Principal**: IAM principal refers to a user, account, service, role, other entity, in terms of evaluating authorization, a principal is defined as the entity that is allowed or denied access to a resource.



Policies

- IAM policies define permissions for an action regardless of the method that you use to perform the operation.

Policy types

- **Identity-based policies**: attach managed and inline policies to IAM identities (users, groups to which users belong, or roles). Identity-based policies grant permissions to an identity
- **Resource-based policies**: attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies. Resource-based policies grant permissions to a principal entity that is specified in the policy. Principals can be in the same account as the resource or in other accounts
- **Permissions boundaries**: use a managed policy as the permissions boundary for an IAM entity (user or role). That policy defines the maximum permissions that the identity-based policies can grant to an entity, but does not grant permissions. Permissions boundaries do not define the maximum permissions that a resource-based policy can grant to an entity
- **Organizations SCPs**: Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). The SCP maximum applies to principals in member accounts, including each AWS account root user. If an SCP is present, identity-based and resource-based policies grant permissions to principals in member accounts only if those policies and the SCP allow the action. If both a permissions boundary and an SCP are present, then the boundary, the SCP, and the identity-based policy must all allow the action
- **Access control lists (ACLs)**: use ACLs to control which principals in other accounts can access the resource to which the ACL is attached. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document structure. ACLs are cross-account permissions policies that grant permissions to the specified principal entity. ACLs cannot grant permissions to entities within the same account
- **Session policies**: pass advanced session policies when you use the AWS CLI or AWS API to assume a role or a federated user. Session policies limit the permissions that the role or user's identity-based policies grant to the session. Session policies limit permissions for a created session, but do not grant permissions. For more information, see Session Policies
- **Inline policies**: you create and manage, and that are embedded directly into a single user, group, or role. Enforce a strict one-to-one relationship between policy and principal. Avoid the wrong policy being attached to a principal. Ensure the policy is deleted when deleting the principal.

Policies Deep Dive

- Anatomy of a policy: JSON document with `Effect`, `Action`, `Resource`, `Conditions` and `Policy Variables`

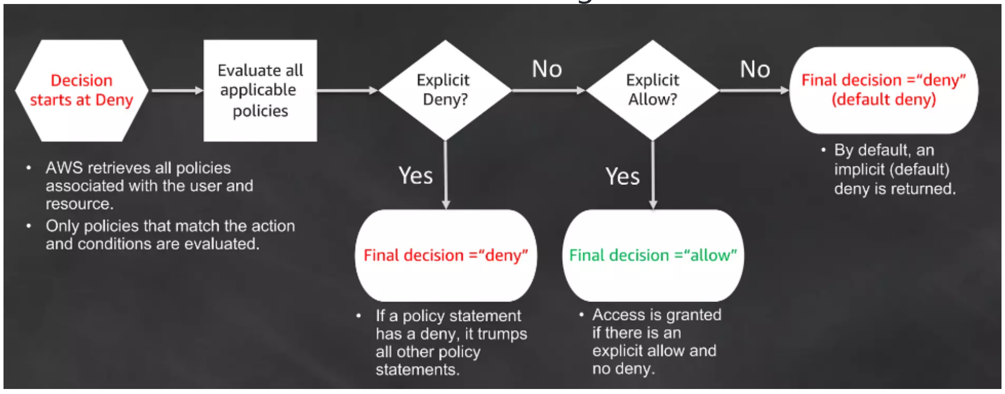
- An explicit `DENY` has precedence over `ALLOW`
- Best practice: use least privilege for maximum security
    - Access Advisor: a tool for seeing permissions granted and when last accessed
    - Access Analyzer: used of analyze resources shared with external entities
- Common Managed Policies:
    - `AdministratorAccess`
    - `PowerUserAccess`: does not allow anything regarding to IAM, organizations and account (with some exceptions), otherwise similar to admin access
- IAM policy condition:
- `"Condition": {`
- `    "{condition-operator}": {`
- `        "{condition-key}": "{condition-value}"`
- `    }`
- `}`
- Operators:
    - String: `StringEquals`, `StringNotEquals`, `StringLike`, etc.
    - Numeric: `NumericEquals`, `NumericNotEquals`, `NumericLessThan`, etc.
    - Date: `DateEquals`, `DateNotEquals`, `DateLessThan`, etc.
    - Boolean
    - IpAddress/NotIpAddress:
        - `"Condition": {"IpAddress": {"aws:SourceIp": "192.168.0.1/16"}}`
    - ArnEquals/ArnLike
    - Null
        - `"Condition": {"Null": {"aws:TokenIssueTime": "192.168.0.1/16"}}`
- Policy Variables and Tags:
    - `${aws:username}`: example `"Resource:["arn:aws:s3:::mybucket/${aws:username}/*"]`
    - AWS Specific:
        - `aws:CurrentTime`
        - `aws:TokenIssueTime`
        - `aws:PrincipalType`: indicates if the principal is an account, user, federated or assumed role
        - `aws:SecureTransport`
        - `aws:SourceIp`
        - `aws:UserId`
    - Service Specific:
        - `ec2:SourceInstanceARN`
        - `s3:prefix`
        - `s3:max-keys`
        - `sns:EndPoint`

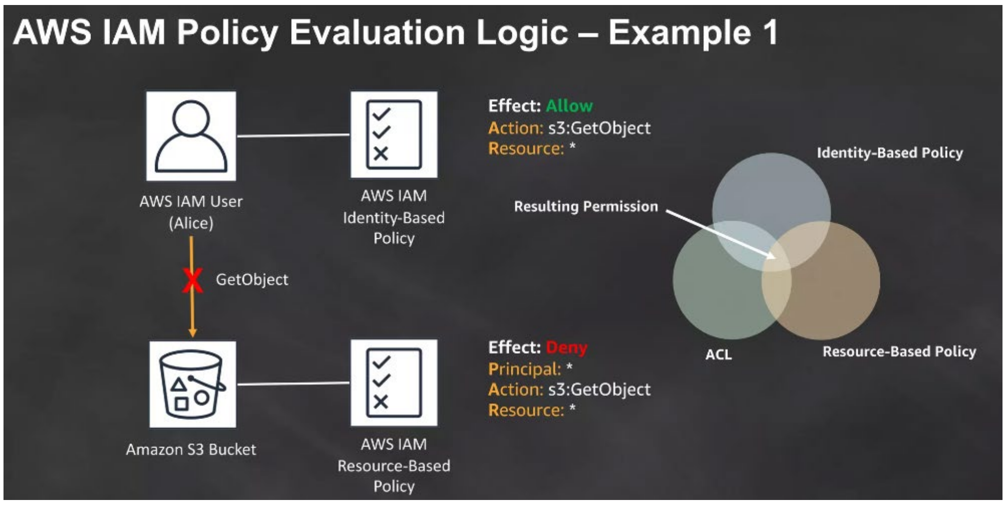
        - `sns:Protocol`
    - Tag Based:
        - `iam:ResourceTag/key-name`
        - `iam:PrincipalTag/key-name`

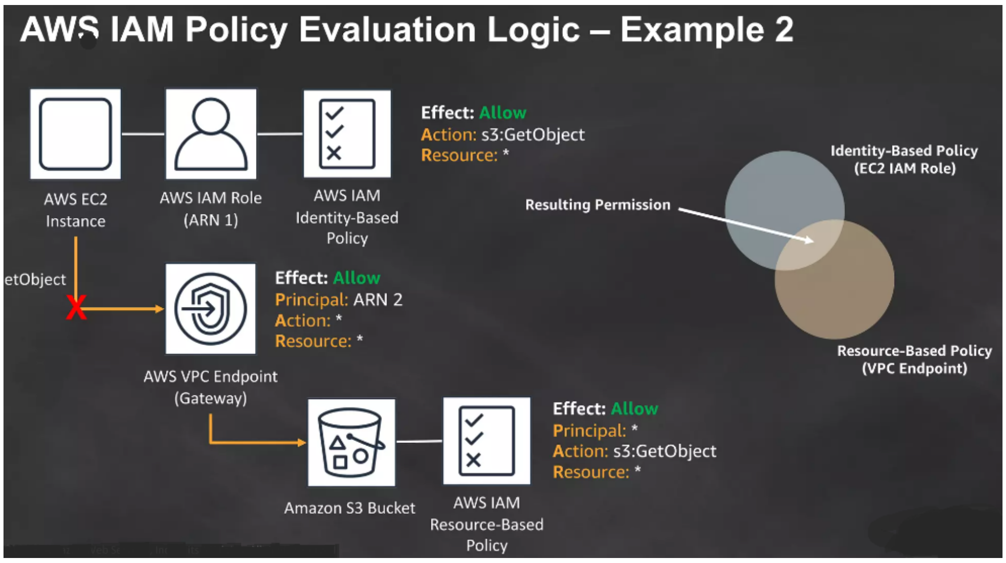
https://docs.aws.amazon.com/zh_cn/IAM/latest/UserGuide/reference_policies_elements.html

Priority Levels in IAM:

- **Explicit Deny**: Denies access to a particular resource and this ruling cannot be overruled.
- **Explicit Allow**: Allows access to a particular resource so long as there is not an associated Explicit Deny.
- **Default Deny (or Implicit Deny)**: IAM identities start off with no resource access. Access instead must be granted


Decision starts at Deny
Evaluate all applicable policies
Explicit Deny?
No
Explicit Allow?
No
Final decision ="deny" (default deny)
• AWS retrieves all policies associated with the user and resource.
• Only policies that match the action and conditions are evaluated.
Yes
Yes
• By default, an implicit (default) deny is returned.
Final decision ="deny"
Final decision ="allow"
• If a policy statement has a deny, it trumps all other policy statements.
• Access is granted if there is an explicit allow and no deny.





AWS IAM Policy Evaluation Logic – Example 2

Effect: Allow
Action: s3:GetObject
Resource: *

Effect: Allow
Principal: ARN 2
Action: *
Resource: *

Effect: Allow
Principal: *
Action: s3:GetObject
Resource: *

AWS EC2 Instance
AWS IAM Role (ARN 1)
AWS IAM Identity-Based Policy
etObject
AWS VPC Endpoint (Gateway)
Amazon S3 Bucket
AWS IAM Resource-Based Policy
Resulting Permission
Identity-Based Policy (EC2 IAM Role)
Resource-Based Policy (VPC Endpoint)

IAM Roles vs Resource Based Policies

- When we assume a role (user, application or service), we give up our original permission and take the permission assigned to the role
- When using a resource-based policy, principal does not have to give up any permissions
- Example: user in account A needs to scan a DynamoDB table in account A and dump it in an S3 bucket in account B. In this case if we assume a role in account B, we won't be able to scan the table in account A

Best practices

- One IAM User per person **ONLY**
- One IAM Role per Application
- IAM credentials should **NEVER** be shared
- Never write IAM credentials in your code. **EVER**
- Never use the ROOT account except for initial setup
- It's best to give users the minimal amount of permissions to perform their job

https://docs.aws.amazon.com/zh_cn/cli/latest/userguide/getting-started-install.html