AWS Identity and Access Management (IAM)

Introduction

- AWS Identity and Access Management (IAM) is a service provided by Amazon Web Services (AWS) that allows you to manage access to AWS resources securely.
- With IAM, you can create and manage users, groups, and roles to control who can access your AWS resources and what actions they can perform.
- IAM enables you to set up fine-grained permissions and policies, ensuring that only authorized users and services can interact with your AWS resources.

Components of AWS IAM

- **Users:** IAM users represent individuals or entities who need access to AWS resources. Each user has their own credentials for authentication. You can attach policies to users to provide permissions
- **Groups:** Groups are collections of IAM users. You can attach policies to groups, making it easier to manage permissions for multiple users with similar roles.
- **Roles:** IAM roles are used by AWS services, applications, or EC2 instances to securely access other AWS resources. Roles are not associated with a specific user or group and are assumed as needed.
- **Policies:** IAM policies are JSON documents that define permissions. You can attach policies to users, groups, or roles to specify what actions they can perform on which resources.
- **Permissions:** Permissions define the actions a user, group, or role is allowed or denied to perform on AWS resources.
- **Access Keys:** IAM users can have access keys (Access Key ID and Secret Access Key) that allow programmatic access to AWS services. These are often used for API calls and command-line tools.

Principle of least privilege (POLP)

- Principle of Least Privilege (POLP) ensures that users and services only have the access they require to perform their tasks.
- Nothing more, Nothing less.
- It helps protect your AWS environment from unauthorized access and misuse.

IAM Users

- IAM users in Amazon Web Services (AWS) are entities that represent individuals, applications, or services that interact with your AWS resources.

- These users have their own credentials (username and password or access keys) that are used for authentication when accessing AWS services and resources.

Here are some key points about IAM users:

1. User Authentication**: IAM users can be authenticated using either AWS Management Console login (username and password) or programmatic access (Access Key ID and Secret Access Key) for API calls and command-line tools.
2. Management: As an AWS account owner, you can create, modify, and delete IAM users as needed to control who has access to your AWS resources.
3. Permissions: IAM users don't have any permissions by default. You need to attach IAM policies to them to specify what actions they can perform and on which AWS resources.
4. Groups: To simplify permission management, you can organize IAM users into groups. Groups allow you to attach policies to a collection of users rather than attaching policies individually to each user.
5. Access Keys: IAM users can have access keys (Access Key ID and Secret Access Key) for programmatic access to AWS services. These keys are used when making API calls and are essential for automation and script-based interactions with AWS.
6. Multi-Factor Authentication (MFA): You can enable MFA for IAM users to add an extra layer of security to their AWS accounts. MFA requires users to provide a secondary authentication in addition to their password.

IAM Groups

AWS Identity and Access Management (IAM) groups are collections of IAM users. They make it easier to manage permissions for multiple users who have similar roles or responsibilities within your AWS environment.

Here are some key points about IAM groups:

1. Simplifies Permission Management: Instead of attaching policies individually to each IAM user, you can create IAM groups and attach policies to the group.
2. Logical Grouping: IAM groups allow you to logically group users who have similar job functions, such as "Developers," "Administrators," or "Finance Team."
3. You can add users to the appropriate IAM group with pre-defined permissions, reducing the manual effort of configuring individual permissions.
4. Policy Attachment: IAM policies are attached to groups to define what actions and resources the members of the group are allowed or denied access to.
5. Users can be added or removed from groups at any time.
6. Group Members: IAM users can be members of multiple IAM groups. For example, a user might belong to a "Developers" group and a "DevOps" group, each with its set of permissions.

IAM Roles

AWS Identity and Access Management (IAM) roles are a fundamental component of AWS security that allow you to securely delegate permissions and access to AWS resources, both within your AWS account and across accounts. Here are key aspects of IAM roles:

1. Temporary Permissions: IAM roles provide temporary permissions. Unlike IAM users or groups, roles are not associated with specific credentials (like passwords or access keys). Instead, users, services, or applications assume a role to temporarily inherit its permissions.
2. Service Roles: AWS services, such as AWS Lambda or EC2 instances, can assume roles to access other AWS resources securely. This allows services to perform actions on your behalf without exposing long-term access keys or credentials.
3. IAM Policies: IAM roles are associated with IAM policies that define the permissions they grant. Policies can be inline policies or managed policies attached to the role. These policies determine what actions can be taken on which AWS resources.

IAM Policies

AWS Identity and Access Management (IAM) policies are JSON documents that define the permissions and access control rules for users, groups, or roles within your AWS account. IAM policies determine what actions are allowed or denied on AWS resources. Here are some key aspects of IAM policies:

1. Permissions Definition**: IAM policies specify the permissions granted or denied for AWS resources. These permissions can include actions (e.g., `s3:PutObject`), resources (e.g., an S3 bucket ARN), and conditions that further refine access.
2. JSON Format: IAM policies are written in JSON (JavaScript Object Notation) format. The JSON structure includes elements like "Effect" (which can be "Allow" or "Deny"), "Action" (the list of allowed or denied actions), "Resource" (the AWS resource to which the policy applies), and optional "Condition" statements.
3. Inline and Managed Policies**: IAM policies can be attached directly to IAM users, groups, or roles as inline policies. Alternatively, they can be created separately as managed policies and attached to multiple users, groups, or roles. Managed policies offer centralized management and can be versioned and shared across accounts.
4. Wildcard (*) Permissions: IAM policies can use wildcards (*) to match multiple actions or resources. For example, you can specify `s3:List*` to allow all list-related actions in Amazon S3.

Structure of IAM Policy

The structure of an AWS Identity and Access Management (IAM) policy is defined in JSON (JavaScript Object Notation) format and consists of various elements that specify the permissions and access control rules. Here's the basic structure of an IAM policy:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow or Deny",
   "Action": "AWS Service Actions",
   "Resource": "AWS Resource ARN or Resource ARN Pattern",
   "Condition": {
    "ConditionKey": {
     "Operator": "Condition Operator",
     "Value": "Condition Value"
    }
   }
  },
  {
   "Effect": "Allow or Deny",
   "Action": "AWS Service Actions",
   "Resource": "AWS Resource ARN or Resource ARN Pattern",
   "Condition": {
    "ConditionKey": {
     "Operator": "Condition Operator",
     "Value": "Condition Value"
    }
   }
  },
  // Additional statements as needed
 ]
}
```

Here's a breakdown of the elements within an IAM policy:

- Version**: Specifies the version of the policy language. The value should be set to `"2012-10-17"` for the current version of IAM policies.

- Statement: An array of one or more policy statements. Each statement defines a set of permissions and access controls. Multiple statements can be included in a policy to cover different scenarios or conditions.
- Effect: Determines whether the statement allows or denies access. It can be set to `"Allow"` or `"Deny"`. An "Allow" statement grants the specified permissions, while a "Deny" statement explicitly denies them.
- Action: Specifies the AWS service actions that are allowed or denied. These actions are typically in the format `"service:Action"`, such as `"s3:PutObject"` or `"ec2:DescribeInstances"`. You can use wildcards (*) to match multiple actions, like `"s3:List*"` to match all list-related actions in Amazon S3.
- Resource: Defines the AWS resource or resources to which the statement applies. Resources are identified using Amazon Resource Names (ARNs) or ARN patterns. For example, `"arn:aws:s3:::example-bucket/*"` specifies all objects within an Amazon S3 bucket.
- Condition: Optional element that allows you to specify conditions under which the statement is applied. Conditions can be based on various factors, such as IP address, date, or resource tags. Conditions are evaluated before granting or denying access.
    - Condition Key: Specifies the condition key (variable) to which the condition is applied.
    - Operator: Defines the condition operator, such as `"StringEquals"`, `"IpAddress"`, or `"DateGreaterThan"`.
    - Value: Specifies the value or values against which the condition is evaluated.

IAM Policy Examples

Allowing Access to an S3 Bucket

This policy allows an IAM user to list the contents of a specific S3 bucket and read objects within it.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "s3:ListBucket",
    "s3:GetObject"
   ],
   "Resource": [
    "arn:aws:s3:::example-bucket",
```

```
        "arn:aws:s3:::example-bucket/*"
      ]
     }
    ]
}
```

Denying Access to Delete S3 Buckets*

This policy denies a user the ability to delete S3 buckets.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": "s3:DeleteBucket",
   "Resource": "*"
  }
 ]
}
```

Allowing Lambda Functions to Access DynamoDB

This policy allows AWS Lambda functions to read and write to a specific DynamoDB table.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
   ],
   "Resource": "arn:aws:dynamodb:us-east-1:12345678901:table/MyTable"
  }
 ]
}
```

Conditional Access Based on IP Address

This policy allows access to an AWS resource conditionally based on the source IP address.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::example-bucket/*",
   "Condition": {
    "IpAddress": {
     "aws:SourceIp": "192.168.1.0/24"
    }
   }
  }
 ]
}
```

IAM - MFA

- Multi-Factor Authentication (MFA) in AWS Identity and Access Management (IAM) adds an extra layer of security to your AWS accounts by requiring users to present two or more separate factors of authentication before they can access AWS resources.
- This helps protect against unauthorized access, even if a user's password is compromised.
- Here's how MFA works in AWS IAM:
  - ✓ Factors of Authentication: MFA in AWS typically involves two factors:
    - Something You Know: This is the user's password.
    - Something You Have: This is a physical or virtual MFA device, such as a hardware token or a virtual MFA app on a smartphone.
  - ✓ Enabling MFA: To use MFA in AWS IAM, a user must enable it for their own account. Users can associate an MFA device with their IAM user account.
  - ✓ Types of MFA Devices:
    - Hardware MFA Device: This is a physical device, often in the form of a keychain, that generates a time-based one-time password (TOTP). AWS provides a list of supported hardware MFA devices.
    - Virtual MFA Device: This is typically a mobile app, such as Google Authenticator or AWS Virtual MFA, that generates TOTP codes. Users can scan a QR code to set up the virtual MFA device.

- Using MFA for Console Login**: Once MFA is enabled, users are required to enter both their password and the current MFA code when logging into the AWS Management Console.

AWS Access Keys

- AWS Access Keys are credentials used for programmatic access to AWS services and resources.
- They consist of two parts: an Access Key ID and a Secret Access Key.
- These keys are associated with an AWS Identity and Access Management (IAM) user or AWS service and are used to make authenticated API requests and command-line tool interactions with AWS services.
- Here are some key points about AWS Access Keys:
    - Access Key ID: The Access Key ID is a public identifier that is used to uniquely identify the AWS user or service making the request.
    - Secret Access Key**: The Secret Access Key is a private key that must be kept confidential. It is used to sign API requests, providing authentication and ensuring the integrity of requests.
    - Use Cases: Access Keys are typically used in the following scenarios:
        - AWS CLI: You can configure the AWS Command Line Interface (CLI) to use Access Keys for programmatic access to AWS services.
        - SDKs and Development: Developers use Access Keys with AWS Software Development Kits (SDKs) and libraries to build applications that interact with AWS services.

AWS CLI

The AWS Command Line Interface (AWS CLI) is a command-line tool provided by Amazon Web Services (AWS) that allows users to interact with various AWS services and resources from a terminal or command prompt.

Here are some key features and concepts related to the AWS CLI:

- Installation: You can install the AWS CLI on various operating systems, including Windows, macOS, and Linux. The installation process typically involves downloading and running an installer or using package.
- Configuration: After installation, you need to configure the AWS CLI with your AWS credentials, including an Access Key ID and a Secret Access Key. You can also configure the default AWS region and output format (e.g., JSON, text).
- Commands and Syntax: The AWS CLI provides a rich set of commands for interacting with AWS services. The general syntax for AWS CLI commands is:

aws <service> <operation> [options and parameters]

For example, to list all Amazon S3 buckets in your account, you would use:

aws s3 ls

Setting up AWS CLI Environment

Setting up the AWS Command Line Interface (CLI) involves several steps, including installation, configuration, and verification. Here's a step-by-step guide on how to set up the AWS CLI:

Step 1: Install the AWS CLI

- Windows: You can download the AWS CLI installer for Windows from the AWS website. Run the installer and follow the installation wizard's instructions.
- macOS and Linux:
  - ✓ On macOS, you can install the AWS CLI using the `brew` package manager:

    brew install awscli

  - ✓ On Linux, you can use `pip` (Python's package manager) to install the AWS CLI. First, ensure you have `pip` installed, and then run:

    pip install awscli --upgrade --user

Step 2: Verify Installation

- To verify that the AWS CLI has been installed successfully, open a terminal or command prompt and run the following command:

  aws --version

  This command should display the AWS CLI version number, confirming that it was installed correctly.

Step 3: Configure AWS CLI

- Configuring the AWS CLI involves specifying your AWS credentials, default region, and output format. You can do this using the `aws configure` command.

  Run the following command and follow the prompts:

  aws configure

  You'll be asked to enter the following information:

- AWS Access Key ID: Enter your AWS Access Key ID.

- AWS Secret Access Key: Enter your AWS Secret Access Key.

- Default region name: Enter the AWS region you want to use as the default (e.g., `us-east-1`).

- Default output format: You can choose the output format for AWS CLI commands, such as `json`, `text`, or `table`.

Step 4: Verify Configuration

- To verify that your AWS CLI configuration is correctly set up, you can run a simple AWS CLI command. For example:

```
aws s3 ls
```

This command lists the S3 buckets in your AWS account. If it successfully lists your buckets without errors, your AWS CLI configuration is working.

AWS Cloudshell

- AWS CloudShell is a browser-based shell environment provided by Amazon Web Services (AWS) that allows you to interact with AWS services and resources directly from your web browser.
- It provides a fully managed command-line interface (CLI) environment preconfigured with AWS CLI, SDKs, and other commonly used tools, making it easy to work with AWS resources without needing to install or configure these tools locally.
- Here are some key features and benefits of AWS CloudShell:
    - Access Anywhere: You can access AWS CloudShell from the AWS Management Console in your web browser, eliminating the need to set up CLI tools on your local machine.
    - Preconfigured Environment: AWS CloudShell comes with the AWS CLI, AWS SDKs, and other commonly used command-line tools preinstalled and preconfigured. This means you can start using AWS services immediately without any setup.
    - No Additional Costs: There is no additional cost for using AWS CloudShell; you only pay for the underlying AWS resources and storage, if applicable.
    - Persistent Storage: AWS CloudShell provides a persistent home directory (5 GB) for storing scripts, configuration files, and other data across sessions.
    - Authentication: AWS CloudShell is automatically authenticated with your AWS credentials from the AWS Management Console. You don't need to configure credentials separately.

- Linux Environment: AWS CloudShell is based on a Linux environment, so you can use Linux commands and utilities in addition to AWS-specific commands.
- Integration: AWS CloudShell integrates with AWS services, making it easy to perform tasks like managing S3 buckets, launching EC2 instances, and configuring AWS resources.
- CloudTrail Logging: AWS CloudShell activity is logged by AWS CloudTrail, providing visibility into actions performed in the environment.

- To access AWS CloudShell:
  - Sign in to the AWS Management Console.
  - Navigate to the AWS CloudShell service from the AWS Console.
  - Click the "Open AWS CloudShell" button.

IAM Security Tools

- **IAM Credentials Report**
  - ✓ It can be used to generate and download a report that lists all your account's users and the status of their various credentials such as password, Access Key, MFA devices etc.
  - ✓ Steps to generate Credentials report:
    - AWS Management Console -> IAM Service -> Credentials Report
- **IAM Access Advisors**
  - ✓ It shows the service permissions granted to a user and when those services were last accessed.
  - ✓ Steps to generate Credentials report:
    - AWS Management Console -> IAM Service -> Users -> Select User
    - In the third tab, we see access advisors

IAM Best Practices

- Don't use the root account except AWS account setup
- Assign users to groups and assign permissions to groups
- Create a Strong Password policy
- Use and enforce the use of multi-factor authentication
- Create and use roles for giving permissions to AWS services
- Use access keys for programmatic access (CLI/SDK)
- Audit permissions of your account with the IAM Credentials Report
- Audit the services accessed by a IAM user with the IAM Access Advisor
- Never share IAM users Login credentials and access keys