EC2 Instance Metadata Service (IMDS)

Overview:
The EC2 Instance Metadata Service (IMDS) is a feature that allows EC2 instances to access information about themselves. This service is extremely useful for obtaining various details about the instance without needing an IAM role specifically for this purpose.

Accessing Metadata:

- **URL**: The metadata can be accessed via a specific URL: `http://169.254.169.254`.
- **Information Available**: From this metadata URL, you can retrieve information such as:
  - Instance ID
  - Instance type
  - Public IP address
  - Private IP address
  - AMI ID
  - IAM Role name
  - Instance tags
  - Security groups

Additionally, you can obtain temporary credentials associated with the IAM role assigned to the instance, though you cannot directly view the IAM policies attached to the role.

User Data vs Metadata:

- **Metadata**: Refers to information about the instance.
- **User Data**: Refers to the launch script executed when the instance starts.

IMDS Versions:

- **IMDSv1**:
  - Accessing metadata is straightforward. You simply make a request to the metadata URL and receive the data.
  - Example command: `curl http://169.254.169.254/latest/meta-data/`
- **IMDSv2**:
  - Introduced for enhanced security.
  - Requires a two-step process to access metadata:
    1. **Obtain a Session Token**: You need to make a PUT request to get a session token.
    2. **Use the Session Token**: Pass the token as a header in subsequent requests to retrieve metadata.
  - **Steps**:
    1. Obtain the session token:

```
TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -
H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
```

2. Use the token to access metadata:

```
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/"
```

Usage:

- **Configuration Scripts**: You can use the metadata service in configuration scripts to dynamically obtain instance-specific details.
- **Automation**: Automate instance-specific settings and configurations by fetching metadata.

Security Considerations:

- **IMDSv2 Adoption**: Due to its enhanced security features, using IMDSv2 is recommended.
- **Least Privilege Principle**: Ensure that IAM roles and policies follow the principle of least privilege, even though metadata service allows access to certain temporary credentials.

Conclusion:
The EC2 Instance Metadata Service is a powerful tool that enables instances to self-discover information and configure themselves dynamically. With the introduction of IMDSv2, AWS has enhanced the security of accessing this metadata, making it a preferred choice for managing EC2 instance information securely.

Practicing with the EC2 Instance Metadata Service (IMDS)

Setting Up the EC2 Instance:

1. **Creating the Instance:**
    - Launch an EC2 instance using the Amazon Linux 2023 AMI.
    - In the advanced details section, note that you can select the metadata version. For Amazon Linux 2023, IMDSv2 is enforced by default.
2. **Security Group and IAM Profile:**
    - Create a security group allowing SSH from anywhere.
    - For this demonstration, no IAM instance profile is initially selected.
3. **Launching the Instance:**
    - Proceed with launching the instance and connect to it using EC2 Instance Connect.

Querying the Metadata Service:

1. **Attempt IMDSv1:**
    - Using EC2 Instance Connect, attempt to query the metadata using IMDSv1:

    ```
    curl http://169.254.169.254/latest/meta-data/
    ```

    - Expect a `401 Unauthorized` error because IMDSv1 is not supported by default on Amazon Linux 2023.
2. **Using IMDSv2:**
    - First, obtain a session token:

    ```
    TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
    ```

    - Verify the token retrieval:

    ```
    echo $TOKEN
    ```

    - Use the token to access metadata:

```
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/"
```

  - Retrieve specific metadata (e.g., hostname and local IPv4):

```
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/hostname"
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/local-ipv4"
```

Accessing IAM Role Credentials:

1. **Attach IAM Role:**
   - Attach an IAM role to the instance through the EC2 console under `Actions > Security > Modify IAM role`.
2. **Querying IAM Role Credentials:**
   - After attaching the IAM role, query the metadata for IAM role credentials:

```
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/iam/security-credentials/"
```

   - Retrieve specific credentials for the IAM role:

```
curl -H "X-aws-ec2-metadata-token: $TOKEN"
"http://169.254.169.254/latest/meta-data/iam/security-
credentials/<role-name>"
```

   - The response includes the access key ID, secret access key, session token, and expiration date.

Summary:

- **IMDSv2**: Amazon Linux 2023 enforces IMDSv2 by default for enhanced security.
- **Metadata Retrieval**: Obtain a session token and use it to query the metadata.
- **IAM Role Credentials**: Attach an IAM role to the instance and query for temporary credentials through the metadata service.

This practice demonstrates the importance of IMDSv2 for secure access to instance metadata and IAM role credentials, illustrating how instances can self-discover and manage configurations dynamically.

Managing Multiple AWS Accounts with Profiles

When working with multiple AWS accounts, using profiles is an efficient way to manage different sets of credentials. Here's how to set up and use profiles with the AWS CLI:

Initial Configuration

1. **Default Configuration:**
    - Typically, the AWS CLI is configured with default credentials found in the `~/.aws/credentials` file under the `[default]` section.
    - The default configuration links to a single AWS account.

Setting Up a New Profile

1. **Configure a New Profile:**

    - To set up a new profile, use the `aws configure --profile <profile-name>` command.
    - Example:

    ```
    aws configure --profile my-other-aws-account
    ```

    - You will be prompted to enter the AWS access key ID, secret access key, default region name, and output format for the new profile.
    - Example input (dummy data for illustration):

    ```
    AWS Access Key ID [None]: <your-new-access-key-id>
    AWS Secret Access Key [None]: <your-new-secret-access-key>
    Default region name [None]: us-west-2
    Default output format [None]: json
    ```

2. **Credential Files Update:**

    - **`~/.aws/credentials` File:**

    ```
    [default]
    aws_access_key_id = <default-access-key-id>
    aws_secret_access_key = <default-secret-access-key>

    [my-other-aws-account]
    aws_access_key_id = <your-new-access-key-id>
    aws_secret_access_key = <your-new-secret-access-key>
    ```

- `~/.aws/config` **File:**

  ```
  [default]
  region = <default-region>
  output = json

  [profile my-other-aws-account]
  region = us-west-2
  output = json
  ```

Using Profiles

1. **Execute Commands with a Profile:**
   - By default, AWS CLI commands use the default profile:

     ```
     aws s3 ls
     ```

   - To use a specific profile, add the `--profile` flag to your command:

     ```
     aws s3 ls --profile my-other-aws-account
     ```

Summary

- **Profiles** allow you to switch between multiple AWS accounts by maintaining separate configurations.
- **Configuration Command:**
  - `aws configure --profile <profile-name>` to set up a new profile.
- **Using Profiles in Commands:**
  - Use `--profile <profile-name>` with any AWS CLI command to specify which profile to use.

By following these steps, you can efficiently manage and switch between multiple AWS accounts, ensuring that your commands target the correct AWS environment.

Using Multi-Factor Authentication (MFA) with AWS CLI

To use Multi-Factor Authentication (MFA) with the AWS CLI, you need to create a temporary session using the `STS GetSessionToken` API. This process involves generating temporary credentials that require a code from your MFA device.

Steps to Set Up and Use MFA with AWS CLI

1. **Assign an MFA Device:**
    - Go to the IAM management console.
    - Select your user and navigate to the "Security credentials" tab.
    - Manage your MFA device by assigning a virtual MFA device (e.g., using the Authy application).
    - Scan the QR code using your MFA app and enter two consecutive MFA codes to complete the setup.
    - Note the ARN (Amazon Resource Name) of the assigned MFA device.
2. **Generate Temporary Session Token:**
    - Use the `aws sts get-session-token` command with the following parameters:
        - `--serial-number`: The ARN of your MFA device.
        - `--token-code`: The current code from your MFA device.
    - Example command:

      ```
      aws sts get-session-token --serial-number arn:aws:iam::<account-id>:mfa/<device-name> --token-code <mfa-code>
      ```

    - This command returns temporary credentials:
        - `AccessKeyId`
        - `SecretAccessKey`
        - `SessionToken`
        - `Expiration`
3. **Configure a New AWS CLI Profile with Temporary Credentials:**
    - Create a new profile using `aws configure --profile <profile-name>`.
    - Example:

      ```
      aws configure --profile mfa
      ```

    - Enter the temporary `AccessKeyId` and `SecretAccessKey` obtained from the `get-session-token` command.

   - Set the default region and output format as needed.

4. **Add the Session Token to the Credentials File:**

   - Open the `~/.aws/credentials` file using a text editor.
   - Add the `aws_session_token` for the newly created profile:

     ```
     [mfa]
     aws_access_key_id = <temporary-access-key-id>
     aws_secret_access_key = <temporary-secret-access-key>
     aws_session_token = <temporary-session-token>
     ```

5. **Use the Profile for AWS CLI Commands:**

   - Execute AWS CLI commands using the `--profile` flag with the name of the profile configured with MFA.
   - Example:

     ```
     aws s3 ls --profile mfa
     ```

By following these steps, you can securely use MFA to authenticate AWS CLI commands with temporary credentials. This ensures enhanced security for your AWS operations by leveraging multi-factor authentication.

AWS SDK Overview

The AWS SDK (Software Development Kit) allows you to interact with AWS services directly from your application's code, bypassing the need for the AWS CLI. This enables seamless integration of AWS capabilities into your applications.

What is an SDK?

An SDK is a set of tools, libraries, and documentation that enables developers to create software applications. For AWS, the SDK provides a way to programmatically access AWS services.

Official AWS SDKs

AWS provides SDKs for multiple programming languages, including:

- Java
- .NET
- Node.js
- PHP
- Python (Boto3)
- Go
- Ruby
- C++

Usage of AWS SDK

The SDKs allow you to make API calls to various AWS services such as DynamoDB, Amazon S3, and many others directly from your application code.

Fun Fact

The AWS CLI is written in Python and uses the Boto3 SDK, which is the Python SDK for AWS.

When to Use an SDK

You should use an SDK when you want to integrate AWS services into your application code. For example, you might use the SDK to:

- Upload files to Amazon S3
- Query data from DynamoDB
- Interact with AWS Lambda

Default Region

If you don't specify a region or configure a default region in your SDK settings, `us-east-1` (Northern Virginia) will be chosen by default. This is important to remember as it may be tested in AWS certification exams.

Practical Application

Throughout your development, especially when working with AWS Lambda functions, you will see how the SDK works in practice with code. This hands-on experience will solidify your understanding of how to use the SDK effectively in real-world scenarios.

By understanding and utilizing the AWS SDK, you can build more robust, scalable, and integrated applications that leverage the power of AWS services directly from your application code.

AWS Limits (Quotas) Overview

AWS limits, also known as quotas, fall into two main categories: API Rate Limits and Service Quotas. Understanding these limits is crucial for optimizing application performance and avoiding throttling.

API Rate Limits

Definition:
API Rate Limits specify the maximum number of API calls you can make to a specific AWS service in a given time frame.

Examples:

- **DescribeInstances API (Amazon EC2):** 100 calls per second.
- **GetObject API (Amazon S3):** 5,500 GET requests per second per prefix.

Handling Throttling:

- **Intermittent Errors:** Occur when you exceed API rate limits. AWS throttles the requests, resulting in errors.
- **Exponential Backoff Strategy:** A recommended approach to handle these errors, where retries are delayed with exponentially increasing wait times between each retry.
- **API Throttling Limit Increase:** If your application consistently exceeds API rate limits, you can request an increase from AWS to handle more requests per second.

Service Quotas (Service Limits)

Definition:
Service Quotas define the maximum number of resources you can use for a specific AWS service.

Examples:

- **On-Demand Standard Instances:** You can run up to 1,152 virtual CPUs.

Increasing Service Quotas:

- **Service Limit Increase:** If you need more resources, you can request a service quota increase by opening a support ticket or using the Service Quotas API for programmatic requests.

Exponential Backoff Strategy

When to Use:

- **ThrottlingException:** Indicates that you are making too many API calls, triggering AWS's throttling mechanism.

AWS SDKs:

- The AWS SDKs automatically implement the retry mechanism with exponential backoff.

Custom Implementations:

- If you are making API calls directly (without using an SDK), you need to implement exponential backoff manually.

Errors to Retry:

- **Server Errors (5XX):** Retry these errors, as they indicate server-side issues that may resolve with time.
- **Client Errors (4XX):** Do not retry these errors, as they indicate issues with the request itself that will not resolve with retries.

How Exponential Backoff Works:

1. **First Retry:** Wait for 1 second.
2. **Second Retry:** Wait for 2 seconds (double the previous wait time).
3. **Third Retry:** Wait for 4 seconds (double the previous wait time).
4. **Subsequent Retries:** Continue doubling the wait time (e.g., 8 seconds, 16 seconds).

Purpose:

- The exponential increase in wait time helps reduce the load on the server by spacing out the retries, giving the server a better chance to recover and handle incoming requests.

Summary

Understanding and managing AWS limits is crucial for ensuring your applications run smoothly. API Rate Limits and Service Quotas define the boundaries of how you can interact with AWS services. When facing throttling, use the Exponential Backoff strategy to manage retries effectively and avoid overloading the server. For consistent high usage, consider requesting an increase in API throttling limits or service quotas to meet your application's needs.

AWS Credentials Provider Chain

When using AWS CLI or SDKs, it is crucial to understand how AWS determines which credentials to use. This is governed by a hierarchy known as the credentials provider chain. The chain specifies the order in which AWS looks for credentials. Here's a detailed breakdown:

Order of Credential Search (CLI)

1. **Command Line Options:**
   - Credentials specified directly in the command line options (e.g., `--region`, `--output`, `--profile`, `--access-key-id`, `--secret-access-key`, `--session-token`) take the highest precedence.
2. **Environment Variables:**
   - If not specified in the command line options, the CLI looks for credentials in the environment variables (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`).
3. **CLI Credentials File:**
   - Next, it checks the CLI credentials file, typically located at `~/.aws/credentials`.
4. **CLI Configuration File:**
   - Then, it looks into the CLI configuration file, usually found at `~/.aws/config`.
5. **Container Credentials:**
   - If running on an ECS task, the CLI will look for credentials provided by the ECS container.
6. **Instance Profile Credentials:**
   - Finally, it checks for instance profile credentials provided by the EC2 instance's IAM role.

Order of Credential Search (SDKs)

The order is similar for AWS SDKs, such as the Java SDK:

1. **Java System Properties:**
   - Credentials specified in Java system properties take the highest precedence.
2. **Environment Variables:**
   - Similar to the CLI, the SDK checks for credentials in environment variables (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`).
3. **Default Credential Profile Files:**
   - The SDK then looks at the default credential profile files.

4. **Amazon ECS Container Credentials:**
    - If running on ECS, it looks for credentials provided by the ECS container.
5. **Instance Profile Credentials:**
    - Finally, it checks for instance profile credentials provided by the EC2 instance's IAM role.

Common Scenario

Consider the following scenario to understand how the credential provider chain affects access:

- **Scenario:**
    - You deploy an application on an EC2 instance and use environment variables to set credentials for an IAM user with `S3FullAccess`. This IAM user can access all S3 buckets.
    - You create and assign an IAM role to the EC2 instance with minimal permissions to access only one S3 bucket.
    - Despite the instance profile, the application still accesses all S3 buckets.
- **Reason:**
    - The environment variables set earlier have higher precedence in the credentials provider chain. Therefore, the application uses the IAM user's credentials instead of the EC2 instance profile's role.
- **Solution:**
    - To resolve this, unset the environment variables. This forces AWS to use the instance profile credentials, adhering to the minimal permissions defined for the EC2 instance role.

Best Practices for Managing Credentials

1. **Avoid Hardcoding Credentials:**
    - Never store credentials directly in your code.
2. **Use IAM Roles:**
    - **Within AWS:** Prefer IAM roles for EC2 instances, ECS tasks, and Lambda functions to inherit credentials.
    - **Outside AWS:** Use environment variables or named profiles, similar to how you configure the CLI.

By following these practices, you ensure a secure and efficient way to manage AWS credentials.

Signing AWS API Requests

When making an API request to AWS, the request must be signed so that AWS can authenticate the requester and authorize the request. This signing process uses your AWS credentials, specifically the access key and secret key, to ensure the request's integrity and authenticity.

Signing the Request

To sign your request, you typically use the Signature Version 4 (SigV4) signing process. This involves using your AWS credentials to generate a signature that accompanies your API request. Although the actual process of generating the signature is complex and involves multiple steps, using AWS SDKs or the AWS CLI simplifies this by automatically handling the signing process for you.

Methods of Transmitting the Signature

There are two primary ways to transmit the signature to AWS once it is computed:

1. **Authorization Header:**
   - The signature is included in the `Authorization` header of the HTTP request. This is the standard method used by the AWS CLI and SDKs.

   Example:

   ```
   GET / HTTP/1.1
   Host: example.amazonaws.com
   Authorization: AWS4-HMAC-SHA256 Credential=<access-key>/20210611/us-east-1/service/aws4_request, SignedHeaders=host;x-amz-date, Signature=<signature>
   ```

2. **Query String:**
   - The signature is included in the URL as a query parameter. This is often used for pre-signed URLs, particularly when sharing resources in Amazon S3.

   Example:

   ```
   https://example.amazonaws.com/?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=<access-key>/20210611/us-east-1/service/aws4_request&X-Amz-Date=20210611T000000Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signature=<signature>
   ```

Practical Example with S3

When accessing an object in Amazon S3, such as a file, a pre-signed URL can be generated. This URL includes all necessary information, including the signature, to authorize the request.

Here's a breakdown of a typical pre-signed URL:

- **Security Token:** Temporary security token (if using temporary credentials).
- **Algorithm:** Specifies the signing algorithm, typically `AWS4-HMAC-SHA256`.
- **Date:** The date when the request was signed.
- **Expires:** The duration for which the URL is valid.
- **AMZ Credentials:** Your AWS credentials, including the access key and other identifying information.
- **AMZ Signature:** The actual signature generated using SigV4.

Example of a Pre-Signed URL Breakdown

```
https://example-bucket.s3.amazonaws.com/coffee.jpg?X-Amz-Security-Token=
<token>&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210611T000000Z&X-Amz-
Expires=3600&X-Amz-Credential=<access-key>/20210611/us-east-
1/s3/aws4_request&X-Amz-Signature=<signature>
```

- `X-Amz-Security-Token`: Temporary security token.
- `X-Amz-Algorithm`: Indicates the signing algorithm.
- `X-Amz-Date`: The timestamp of the request.
- `X-Amz-Expires`: How long the URL is valid (in seconds).
- `X-Amz-Credential`: Your access key and credential scope.
- `X-Amz-Signature`: The generated signature.

Key Takeaways

- **SigV4:** Signature Version 4 is used to sign AWS API requests.
- **Authorization Header:** One method to include the signature in the HTTP request.
- **Query String:** An alternative method to include the signature in the URL.
- **Pre-Signed URLs:** Commonly used in S3 to grant temporary access to resources.
- **Automatic Signing:** AWS CLI and SDKs automatically handle the signing process, simplifying API requests.

Understanding these concepts ensures that your API requests to AWS are secure and properly authenticated, following best practices and maintaining compliance with AWS's security protocols.