AWS IAM でのマルチアカウント管理

はじめに

本ページは、AWS に関する個人の勉強および勉強会で使用することを目的に、AWS ドキュメントなどを参照し作成しておりますが、記載の誤り等が含まれる場合がございます。

最新の情報については、AWS 公式ドキュメントをご参照ください。

Contents

はじめに

AWS アカウントでユーザーを管理するには、次の方法があります。

1. 各 AWS アカウントで IAM ユーザーを個別に管理する
2. **1 つのアカウントで IAM ユーザーを一元管理し、他のアカウントにスイッチロールする**
3. AWS IAM Identity Center で管理する

今回は、「2. 1 つのアカウントで IAM ユーザーを一元管理し、他のアカウントにスイッチロールする」について解説します。

マルチアカウント管理の全体像

 overview

マルチアカウント管理では、以下の役割りの AWS アカウントが存在します。

- IAM 「Jump アカウント（IAM ユーザーを一元管理するための中央アカウント）」が 1 つ
  - この AWS アカウントではワークロードを配置しません。
- ワークロードを配置する「メンバーアカウント（個別の環境用アカウント）」が 1 つ以上
  - 例えば、開発環境、検証環境、本番環境といったものがメンバーアカウントに該当します

メリット

- 認証情報の一元管理: Jump アカウントを使用することで、各 AWS アカウントに個別の IAM ユーザーを作成する必要がなくなります。これにより、認証情報の管理が簡素化され、セキュリティリスクが低減します。
- セキュリティの向上: IAM ユーザーを管理する Jump アカウントを使用することで、アクセス権限を集中管理でき、最小権限の原則を適用しやすくなります。また、多要素認証（MFA）を導入することで、セキュリティがさらに強化されます。
- 運用の効率化: 一度 Jump アカウントにログインすれば、各作業対象のアカウントにスイッチロール（AssumeRole）するだけでアクセスできるため、複数のアカウントにログインし直す手間が省けます。

デメリット

- 実装の複雑さ: マルチアカウント管理の構築には、複雑な設定が必要となり、初期の実装工数が大きくなる可能性があります。
- 学習コスト: チームメンバーがこの方式に慣れるまでに時間がかかる可能性があります。

ただし、AWS CDK や CloudFormation などでテンプレートとして作成しておけば、実装工数を大幅に削減することが可能です。また、長期的には運用効率の向上によりこれらのデメリットを相殺できると考えられます。

メンバーアカウントでの設定

① 許可ポリシーの設定

 step1

操作の許可/拒否を定義したポリシーを必要な分だけ作成します。

 step1

以下のような AWS 管理ポリシーを使用する場合は、作成不要です。カスタマイズした権限セットが必要な場合に作成します。

- AdministratorAccess
- PowerUserAccess
- ReadOnlyAccess

AWS CDK(TypeScript)の実装例

※ 本来は変数を使用するところの一部を便宜上、固定値にしています

以下は、EC2 インスタンスと EBS ボリュームの管理権限を持つカスタムポリシーの作成例です。 AWS ドキュメント>Amazon EC2: タグに基づいて EC2 インスタンスに Amazon EBS ボリュームをアタッチまたはデタッチする を参考にしています。

```
const ec2EBSOwner = new iam.ManagedPolicy(this, "EC2EBSOwner", {
  managedPolicyName: [
    "@policy",
    "ec2",
    "ebs",
    "owner",
    props.pjName,
    props.envName,
  ].join("-"),
  statements: [
    new iam.PolicyStatement({
      effect: iam.Effect.ALLOW,
      actions: ["ec2:AttachVolume", "ec2:DetachVolume"],
      resources: ["arn:aws:ec2:*:*:instance/*"],
      condition: {
        StringEquals: { "aws:ResourceTag/Department": "Development" },
      },
    }),
    new iam.PolicyStatement({
      effect: iam.Effect.ALLOW,
      actions: ["ec2:AttachVolume", "ec2:DetachVolume"],
      resources: ["arn:aws:ec2:*:*:volume/*"],
      condition: {
        StringEquals: { "aws:ResourceTag/VolumeUser": "${aws:username}" },
      },
    }),
  ],
});
```

② ロールの作成

 step2

許可ポリシーを紐づけたロールを作成します。

 step2

③ ロールの信頼ポリシー（信頼関係）と使用条件

 step3

 step3

信頼ポリシー（信頼関係）

Jump アカウントのどの IAM ユーザーに対してロールを引き受けることができるか（使用してもよいか）を定義します。 IAM ユーザーの指定は「全て」か「特定のユーザーのみ」といった指定ができます。 また、

信頼ポリシーで記述するアクションは`sts:AssumeRole`のみです。

ここでは、Jump アカウント（999999999999）からのみロールを引き受けることができるシンプルな例を示します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

`"AWS": "arn:aws:iam::999999999999:root"` はルートユーザーという意味ではなく、全ての IAM ユーザー/グループという意味です。 `"AWS": "999999999999"`と指定しても同じ意味になります。（参考：AWS ドキュメント>AWS アカウント プリンシパル）

応用として、「特定のユーザーのみ」という指定も可能です。次の例の場合は、Jump アカウントの「Alice、Bob、Carol」のユーザーを信頼します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:user/Alice",
        "AWS": "arn:aws:iam::999999999999:user/Bob",
        "AWS": "arn:aws:iam::999999999999:user/Carol"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

注意として、以下は指定することができません。

- IAM グループでの指定（参考：AWS ドキュメント>プリンシパルの指定）
- `arn:aws:iam::999999999999:user/*` といったワイルドカードを使用したすべてのユーザー（参考：AWS ドキュメント>IAM ユーザープリンシパル）

使用条件

使用条件を追加することで、さらに細かい制限を設けることができます。使用条件は`Condition`句で指定します。 例えば、MFA を使ったログインの場合のみロールを引き受けることを許可する条件はつぎのようになります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": "true"
        }
      }
    }
  ]
}
```

使用条件の応用

- 送信元 IP アドレスで許可 ①

指定した IP アドレスからのアクセスのみロールの引き受けを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": ["192.168.0.0/16", "172.16.0.0/16"]
        }
      }
    }
  ]
}
```

- 送信元 IP アドレスで許可 ②

① のパターンと同じになりますが、記載方法が異なります。 指定した IP アドレスではないアクセスを拒否します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": ["192.168.0.0/16", "172.16.0.0/16"]
        }
      }
    }
  ]
}
```

- MFA あり かつ IP 制限

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": "true"
        },
        "IpAddress": {
          "aws:SourceIp": ["192.168.0.0/16", "172.16.0.0/16"]
        }
      }
    }
  ]
}
```

この場合、次のようにDenyで分けたほうが分かりやすいかもしれません。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::999999999999:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": ["192.168.0.0/16", "172.16.0.0/16"]
        }
      }
    }
  ]
}
```

AWS CDK(TypeScript)の実装例

※ 本来は変数を使用するところの一部を便宜上、固定値にしています

```
const maxSessionDurationSeconds: cdk.Duration = props.maxSessionDurationSeconds
  ? cdk.Duration.seconds(props.maxSessionDurationSeconds)
  : cdk.Duration.hours(4);

const devAdminRole = new iam.Role(this, "DevAdminRole", {
  roleName: ["@jobrole", "devadmin", props.pjName, props.envName].join("-"),
  assumedBy: new iam.PrincipalWithConditions(
    new iam.AccountPrincipal("999999999999"),
    {
      BoolIfExists: { "aws:MultiFactorAuthPresent": "true" },
    }
  ),
  description: "Development Administrator Role.",
```

```
    maxSessionDuration: maxSessionDurationSeconds,
    managedPolicies: [
      iam.ManagedPolicy.fromAwsManagedPolicyName("AdministratorAccess"),
      :
    ],
  });
```

Jump アカウントでの設定

④ メンバーアカウントのロール使用許可

 step4

Jump アカウントでユーザーが引き受けること(`sts:AssumeRole`)ができるロールを指定したポリシーを作成します。

特定のロール名やワイルドカードによる指定の方法があります。

- 特定のロールのみ

指定されたロールのみ引き受けることができます。

最小権限の原則に従って、必要なもののみ指定する方法を推奨します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["sts:AssumeRole"],
      "Resource": ["arn:aws:iam::111111111111:role/RoleName"],
      "Effect": "Allow"
    }
  ]
}
```

- ワイルドカードで指定

ワイルドカード (*) を使用し、ロール名が文字列 `RolePrefix` で始まるものを引き受けることができます。
※　ベストプラクティスとして、最小権限の原則に従うので非推奨 ※　名前の付け方次第では、想定外のロールに対して知らないうちに引き受け許可してしまうリスクがあります。命名規則など十分に考慮してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["sts:AssumeRole"],
      "Resource": ["arn:aws:iam::111111111111:role/RolePrefix*"],
      "Effect": "Allow"
```

```
    }
  ]
}
```

- 全て

`role/*`とすることで IAM ロールのすべてを引き受けることができます。 ※　ベストプラクティスとして、最小権限の原則に従うので非推奨 ※　メンバーアカウント側で新しいロールを作成すると自動的に引き受けが許可されてしまうリスクがあります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["sts:AssumeRole"],
      "Resource": ["arn:aws:iam::111111111111:role/*"],
      "Effect": "Allow"
    }
  ]
}
```

AWS CDK(TypeScript)の実装例

※ 本来は変数を使用するところの一部を便宜上、固定値にしています

```
// スイッチロール：開発管理者
const assumeRoleDevAminPolicy = new iam.ManagedPolicy(
  this,
  "AssumeRoleDevAminPolicyDev111111111111",
  {
    managedPolicyName: [
      "@jobpolicy",
      "allow",
      "switch",
      "devadmin",
      props.pjName,
      "dev", // メンバーアカウントの環境識別子（dev/stage/test/prod）
      "111111111111", // メンバーアカウントのAWSアカウントID
    ].join("-"),
    description: "Development Administrator Switch Role.(dev:111111111111)",
    statements: [
      new iam.PolicyStatement({
        effect: iam.Effect.ALLOW,
        actions: ["sts:AssumeRole"],
        resources: [
          `arn:aws:iam::111111111111:role/@jobrole-devadmin-${props.pjName}-dev`,
        ],
      }),
    ],
```

```
  }
);
```

利用者への情報提供

利用者には以下の情報を提供します。

- Jump アカウントへのログイン URL（e.g. `https://999999999999.signin.aws.amazon.com/console`）
- ロールの ARN
- メンバーアカウントの AWS アカウント ID（e.g. `111111111111`）

ロールの切り替え方法

 step5

コンソール

AWS ドキュメント>ロールの切り替え (コンソール)を参考にします。

管理者より連絡された情報をもとに、コンソールでロールの切り替えを行います。`Display name ( 表示名 )` は分かりやすい名前にします。表示名を日本語にするとロール履歴で文字化けする場合がありますので、英語での指定を推奨します。 `Display name`を指定しなかった場合は、`<role name> @ <AccountID>` となります。

`Display color`は任意指定ですが、指定することを推奨します。視覚的に認識できるので切り替えミスの軽減が期待できます。 例えば本番環境など注意が必要なアカウントは赤色といったルールを決めるとよいでしょう。

 switch-role

過去に切り替えたことがあるロールは履歴として最大 5 件まで表示されます。

 switch-role-history

この履歴は Cookie の`noflush_awsc-roleInfo`に次の構造で保持しています。 ※Cookie を削除すると、ロール履歴も消えてしまうので注意が必要です。

```
{
    "bn": "IAMユーザー名",
    "ba": "JumpアカウントのAWSアカウントID",
    "rl": [
        {
            "a": "メンバーアカウントのAWS Account ID",
            "r": "IAM role name",
            "d": "Displayname",
            "c": "Display color"
        },
        :
```

```
    ]
}
```

AWS CLI

AWS ドキュメント>ロールの切り替え (AWS CLI)を参考にします。

`.aws/credentials`に以下を定義します。

`Alice`ユーザーが`developerAdmin`ロールに切り替える場合の例です。

```
[xxxx-accesskey]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

[xxxx-prod]
region = ap-northeast-1
role_arn = arn:aws:iam::111111111111:role/developerAdmin
mfa_serial = arn:aws:iam::999999999999:mfa/Alice
source_profile = xxxx-accesskey
```

```
aws s3 ls --profile xxxx-prod
```

次のコマンドを実行して、IAM ロールを引き受けたことを確認します。

```
aws sts get-caller-identity --profile

{
    "UserId": "AROAXXXXXXXXXXXXXXXX:botocore-session-1722942390",
    "Account": "111111111111",
    "Arn": "arn:aws:sts::111111111111:assumed-role/developerAdmin/botocore-
session-1722942390"
}
```

Chrome 拡張機能を利用した効率的なスイッチロール

ブラウザの Cookie を使用したロール履歴では最大で 5 つしか保持できませんが、Chrome 拡張機能を使えば 6 個以上のロールを瞬時に切り替えることができます。

その拡張機能は、AWS Extend Switch Rolesです。Chrome ウェブストアでインストールできます。

拡張機能の設定で次のように指定します。

```
[xxxx]
aws_account_id = 999999999999
```

```
[xxxx-prod]
role_arn = arn:aws:iam::111111111111:role/developerAdmin
source_profile = xxxx
color = 00FF7F
[xxxx-dev]
role_arn = arn:aws:iam::222222222222:role/developerAdmin
source_profile = xxxx
color = 00FF7F
image = "https://example.com/sample.png"
```

ロールの切り替えを CloudTrail で見る

イベント名 = SwitchRoleの条件でイベント履歴を検索することで確認できます。


```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAXXXXXXXXXXXXXXXX:Alice",
    "arn": "arn:aws:sts::111111111111:assumed-role/developerAdmin/Alice",
    "accountId": "111111111111"
  },
  "eventTime": "2024-01-01T00:00:00Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "SwitchRole",
  "awsRegion": "ap-northeast-1",
  "sourceIPAddress": "192.0.2.1",
  "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36",
  "requestParameters": null,
  "responseElements": {
    "SwitchRole": "Success"
  },
  "additionalEventData": {
    "SwitchFrom": "arn:aws:iam::999999999999:user/Alice",
    "RedirectTo": "https://ap-northeast-1.console.aws.amazon.com/console/home?
region=ap-northeast-1#"
  },
  "eventID": "87337348-62a8-4e1f-9c64-79566dc7ca56",
  "readOnly": false,
  "eventType": "AwsConsoleSignIn",
  "managementEvent": true,
  "recipientAccountId": "111111111111",
  "eventCategory": "Management",
  "tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "ap-northeast-1.signin.aws.amazon.com"
  }
}
```

まとめ

本記事では、AWS IAM を使用したマルチアカウント管理の実装方法と利点について解説しました。Jump アカウントを活用することで、セキュリティの向上、運用の効率化、そして最小権限の原則の効果的な適用が可能になります。適切に実装することで、複雑な組織構造においても、AWS リソースへのアクセスを安全かつ効率的に管理できます。ただし、初期の実装には一定の工数が必要なため、組織の規模や要件に応じて導入を検討することをお勧めします。