Introduction to AWS IAM

Abhishek Giri

Software Consultant

Knoldus Software LLP

Agenda

- What is IAM?
- IAM as a service
- IAM Best Practices
- Elements of IAM policy
- UseCase
- Policy Simulator

knólx

What is IAM?

Identity and access management (IAM) is the security discipline that enables the right individuals to access the right resources at the right times for the right reasons.

IAM enables you to securely control access to your application or product services and resources for your users.

Using IAM, you can create and manage users and groups and use permissions to allow and deny their access to the resources.

knólx

AWS: IAM as a service

AWS Identity and Access Management

AWS IAM roles are a web service that gives you secured "Control Access" to AWS services for your users. IAM policies specify which services/actions are allowed or denied.

You attach policies to group, users, and roles, which are then subject to permissions that you define.

With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users can access.

Policies can be granted either from AWS API programmatically or the AWS management console.

IAM gives you following features :

- Shared access to your AWS account.
- Granular permission.
- Secure access to your AWS resources.
- Identity Information.
- Integrated with many AWS resources.
- Free to use.

Types of Policies

- **Managed Policy** - Standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies apply only to identities (users, groups, and roles) - not resources.
- **Inline Policy** - Policies that you create and manage, and that are embedded directly into a single user, group, or role. Resource-based policies are another form of inline policy.

IAM Groups

- A collection of IAM users.
- You assign permission to group, all IAM user in the inherit those permission.

IAM Users

- Can have username/password to login to aws console.
- Can have aws credentials for making API calls to interact with aws services.
- New IAM user have no permission to do anything, permission must be explicilty granted.

IAM Roles / Instance profile

- The permission of an IAM role can be granted/assigned to EC2 instance.
- All AWS Sdk has buit-in way to auto discover AWS credentials on AWS EC2.

  -Credential file

  -Environmental variable

  -Instance profile

IAM Policies

- When you create a IAM group,user you associate an IAM policy with it which specify the permission that you want to grant.
- IAM policies are JSON formatted document that defines AWS permission.

Elements of policy

- Version - Version specifies the current version of the policy language. It must specify it before the statement element. In this case, our version is "2012-10-17."
- Statement - The Statement element is the main element of the policy. This element is required. The Statement element contains an array of individual statements. Each individual statement is a JSON block enclosed in braces { }.
- Effect - The Effect element is required and specifies whether the statement will result in an allow or an explicit deny.
- Action - The Action element describes the specific action or actions that will be allowed or denied. Each AWS service has its own set of actions that describe tasks that you can perform with that service.
- Resource - The Resource element specifies the object or objects that the statement covers. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN.
- Principal – The Principal element specifies the identity. It is use to specify the User,AWS account that is allowed or denied acces to a resource.

Sample JSON

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "s3:ListBucket",
        "Resource": "arn:aws:s3:::example_bucket"
    }
}
```

IAM Best Practices

- Protect the “root” account.
- Create the individual IAM user.
- Create and use groups.
- Set up a strong password policy.
- Use multifator authentication.
- Use Roles/Instance profiles.
- Rotate credentials often.
- Monitor IAM activity.

Protect the “root” account.

- Root account in AWS has full access to any service.
- By design it permission cannot be restricted.
- Never create AWS credentials keys.

Create the individual IAM user.

- Each user get his/her account.
- Makes managing of users easy.
- Makes defining policies of each user easy.

Create and use groups

- Group allows you to logically define set of users.
- Groups can define different set of policies.
- Users can be part of multiple groups. They will inherit permission for both the groups.

Set up a strong password policy.

- Users should have permission to manage their own passwords.
- You can define a strong password policy that enforces things like minimum length, complexity, periodic rotation etc.

Use multifator authentication

- In addition to using a username and password, IAM has an option setting up a second factor.

Use Roles/Instance profiles.

- If you have an app/script that needs to make an API call to AWS, as far as possible avoid using static access keys.
- Instead use Roles/Instance Profiles.
- AWS automatically expires the credentials.

Rotate credentials often

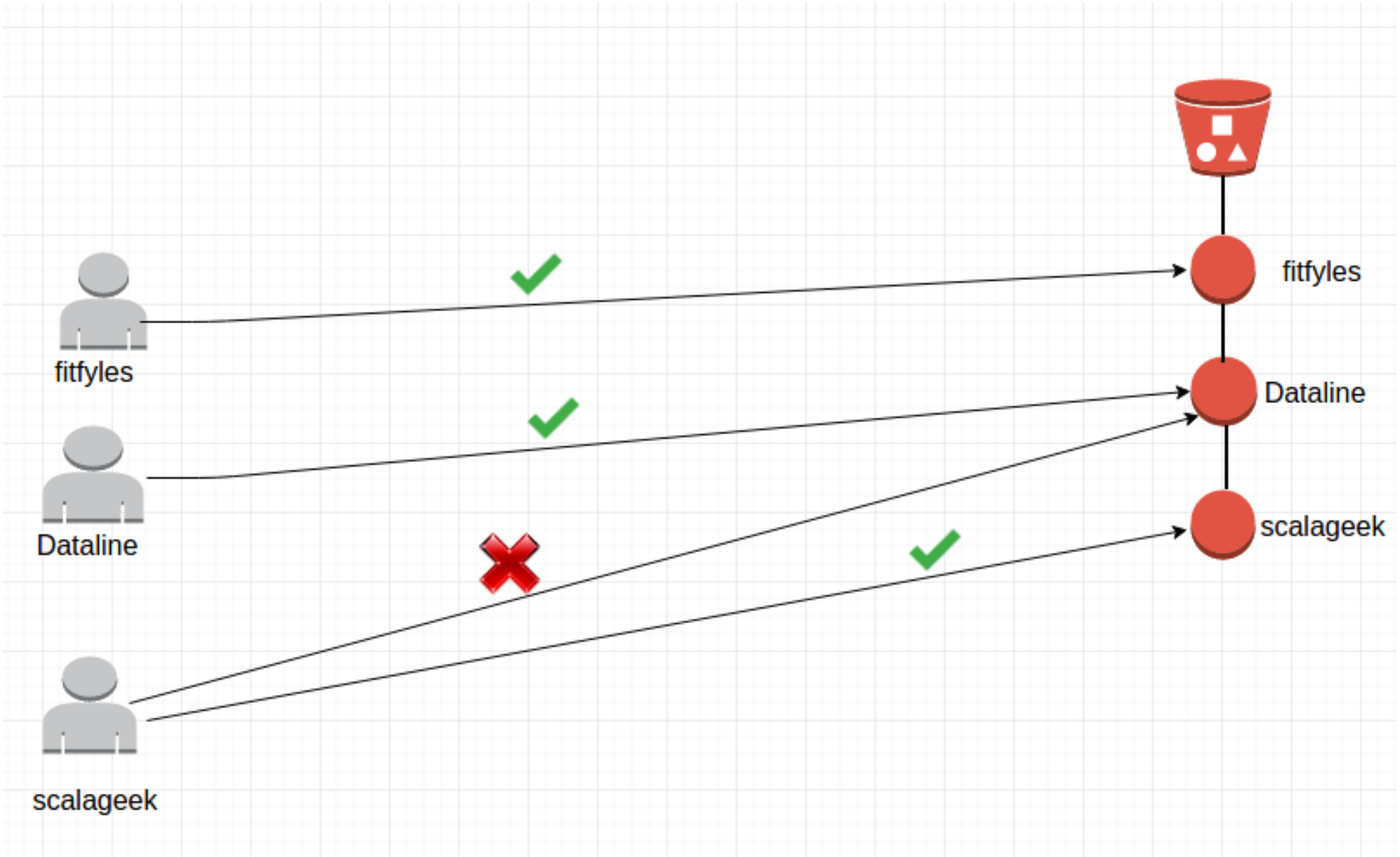
- Enforce all the keys and passwords to be changed often.
- Passwords should be changed once in 90 days.
- Keys could be rotated much more often.
- All keys should have the permission to create another set of keys and delete the old ones.

Monitor IAM activity

- AWS gives you logs for ALL IAM operations.
- Typically this will be in Cloud Trail. Logs are sent to S3 bucket you define.
- Use this information to keep a close watch on what's happening.
- Setup alerts on interesting activities.

UseCase

CodeSquad


fitfyles
Dataline
scalageek
fitfyles
Dataline
scalageek

knólx

In the above diagram, each user has access to his/her object in the bucket.

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": [
"s3:PutObject",
"s3:GetObject",
"s3:GetObjectVersion"
    ],
"Resource": "arn:aws:s3:::examplebucket/${aws:username}/*"
    }]
}
```

Instead of attaching policies to each user, policies can be attached at the group level. After that, we can add users to that group. The following policy allows a set of Amazon S3 permissons in bucketName/${aws:username} folder. When the policy is evaluated, the policy is replaced by requested username.

Policy Simulator

https://policysim.aws.amazon.com/home/index.jsp?#

Reference Link

http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Thank You