AWS IAM

THE GATEKEEPER OF YOUR CLOUD-ECOSYSTEM

INTRODUCTION 👋

For seriously working with AWS, there's no way around AWS' **I**dentity and **A**ccess **M**anagement (**IAM**).
Skipping to understand its core principles will bite you again and again in the future.

So it's worth to take the time to do a deep dive, avoiding frustrations at a later stage of your journey.

PRINCIPALS ⚡

Resource-based policies also need to define a **principal**.

It indicates for which account, (federated) user, or role user you'd like to allow or deny access.

There are more specifics for this type of policy - for example they **aren't** affected by permission boundaries.

LEAST PRIVILEGE 🎯

Least Privilege states that you should always just assign the permissions that are actually needed for the service to fulfill its goals.

💡 Example:
An app that runs on Lambda and now needs access to a recently created DynamoDB table.

🔥 **Bad Solution**: all permissions for complete service.
- action: **dynamodb:***
- resource: *

🌟 **Good solution**: fine-grained access rights.
- restricting actions to use-case, e.g. **dynamodb:Query**
- restricting resources to the resource's **ARN**s

CAPABILITIES 🔶

I'm confident that IAM is AWS' most complete service. It's thoughtfully built, contains a lot of features and integrates perfectly with other services.

- managing users for your AWS account, with individual rights and/or temporary access.
- enforcing password policies or MFA.
- setting rights boundaries for AWS services & your apps.
- identity federation.

... and **much** more.

CREDENTIALS 🔑

For accessing AWS' API to maintain your infrastructure via code, you'll need credentials.

Those can be created at your security credentials page by clicking ***Create Access Key*** to get a set of **Access Key ID** & **Secret Access Key**.

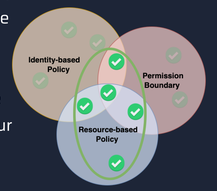
💡 There's no way to display the Secret Access Key again, but you can create a new pair at **any time**.

PERMISSION BOUNDARIES 🔒

Help you to restrict effective permissions for a user or role. They also contain policies that describe actions and resources, but they are acting as an outer boundary.

So if your boundaries only lists **dynamodb:Query** for all resources, a role with **dynamodb:*** can't update or delete items, but only query.

Boundaries can be defined in one place but re-used across all of your account's roles and users.



KEY TERMS 📚

- **User**: end-user, accessing the console or AWS API.
- **Group**: a group of users, sharing the same privileges.
- **Policy**: a defined list of permissions.
- **Role**: a set of policies, that can be assumed by a user or AWS Service to gain all of the policies' permissions.

ROOT USER 👑

It's **not recommended** to work with your root credentials. Those should only be used to enable MFA, create your first IAM user, and should then be locked away securely.

Go to your security page, select ***Manage MFA Device*** and choose your favorite authentication app.
💡 If there's already an Access Key, you can **delete** it as you shouldn't work with your root credentials in any way.

Afterward, you can create your first dedicated user by clicking on **User > Add User**.
You can make use of AWS' predefined policies at first. If you're working on a serious project, make sure to use groups to manage rights across different roles.

TOOLING 🛠️

There's a lot of **support** for following security best practices!

- **AWS Trusted Advisor:** reviews your permissions for unnecessary rights, best practice violations and double checks that you've enabled AWS security features for your services and resources.
- **AWS Policy Simulator:** helps to build, validate and troubleshoot your policies. It supports identity-based, resource-based and even Organizations service control policies.
- **3rd party tools** like Dashbird.io guiding you with security recommendations & well-architected tips.

POLICIES 📝

At AWS IAM, everything revolves around policies.
By default, all actions for all services are **denied**, so you have to explicitly grant permissions by adding a policy with your targeted **actions** and **resources** to your service role, user, or group.

You can attach **one** or **multiple policies** to a role and each policy can contain one or multiple **Statements**.

💡 An important detail about policies: they come in two different **types**:
- attached to a **user, group, or role**.
- attached to a **resource**, e.g. S3 bucket, SQS queue, or KMS key.

ACCESS DENIED ❌

When working with AWS - especially in the beginning - you **will** face this message at least from time to time.

Mostly, AWS' API error response will point you to **missing permissions** so you can easily extend your policy.

Sometimes it's not that easy and you'll need to get back to the documentation.

🔥 What you should **not do**: just blindly extending your policy with all permissions for your target service by adding wildcards for actions & resources.
Example: **action: ["dynamodb:*"]** and **resource: ["*"]**

You won't learn anything by that and you're not respecting the rule of **Least Privilege**.

STATEMENTS 🔑

Statements contain the permissions you want to grant and are defined as JSON.

Let's have a look at what a statement **must** contain:
- **Effect** - the indication if it's a **Deny** or **Allow**.
- **Action** - a list of actions.
- **Resource** - a list of resources for which the actions are granted (can be omitted for resource-based policies).

💡 Extend your policies with **conditions**!

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

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
      "NotIpAddress": {
        "aws:SourceIp": [
          "192.0.2.0/24",
          "203.0.113.0/24"
        ]
      }
    }
  }
}
```

AWS' Command Line Interface 🛠️

With your credentials in place, you should install the AWS CLI. Run **aws configure** at first for doing the initial setup.

You'll be asked for your Access Key ID & Secret Access Key, as well as your **default AWS region** (e.g. **us-east-1**) and CLI **output** (e.g. **json**).

When you're working with different accounts and/or roles and have enabled MFA, it's recommendable to get some **tooling support**.

My favorite daily tools to use:
- **AWSume** - awsu.me
- **Leapp** - leapp.cloud (also supports Azure! 💙)

Both are **easy to set up** and **work** with.

/in/tpschmidt
@tpschmidt_

aws