AWS IAM

SECURELY MANAGE IDENTITIES AND ACCESS TO SERVICES AND RESOURCES

INTRODUCTION 👋

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

It's worth to take the time to deep dive and avoid frustrations at a later stage of your journey.

CAPABILITIES

From our perspective: IAM is the **most complete service** of AWS. It's thoughtfully built and integrates natively with other services while abstracting away many burdens from the past like cumbersome credential management.

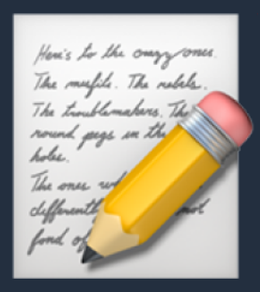
Its feature set contains among others:

- managing users for your AWS account, with individual permissions that can even be temporary.
- enforcement of password policies or MFA.
- permission boundaries for services and applications.
- federation with other identity providers.

KEY TERMS 

IAM comes with multiple core concepts:

- **User**: an identity representing a user accessing AWS.
- **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 the permissions of the policy.

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**.

Policies come in two different **flavours**:

- **Identity-based**: attachable to **users, groups, or roles**.
- **Resource-based**: attachable to **resources**, e.g. S3 buckets, SQS queues, or KMS keys.

STATEMENTS

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

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).

Additionally, you can extend your policies with **conditions** to further drill down permissions.

```
{
  "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"
        ]
      }
    }
  }
}
```

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.

CREDENTIALS 🔑

For accessing the AWS API to create, update or delete your infrastructure resources 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** and **Secret Access Key**.

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

ROOT USER 👑

It's explicitly recommendated by AWS to not work with your root credentials, as it’s permissions are too wide.

First usage recommendation for your root user:

- go to your security page, select **Manage MFA Device** and assign a security-token device, e.g. Google Authenticator.
- if there's already an Access Key, you should delete it as you also shouldn’t work with the CLI.
- create your first dedicated user via **User > Add User**. You can make use of AWS' predefined policies at first.
- lock away your root credentials securely.

AWSFUNDAMENTALS.COM

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 responses will excactly point you the missing permission so you can easily extend your policy.

Sometimes it's not that easy and you'll need to get back to the documentation to find out about required permissions. A great resource is the **Actions, resources, and condition keys for AWS services** with can be found at the **Service Authorization Reference** and does lists every IAM action, resource and conditions for every AWS service.

What you should never do: just blindly extending your policy with all permissions for your target AWS service by adding wildcards for actions and resources, e.g. **action: ["dynamodb:*"]** and **resource: ["*"]**.

You won’t gain any learnings for IAM and you’re distributing unnecessary permissions which can lead to critical incidents.

LEAST PRIVILEGE 

Least Privilege states that you should always only assign the permissions that are actually **required for the service to fulfils its goals**.

Looking at an example: your application runs on Lambda and needs access to a single DynamoDB table.

- **Bad**: assign permissions for DynamoDB via wildcards.
- **Good**: fine-grained permissions, explicitly specifying the actions and resources.

```
data "aws_iam_policy_document" "dynamodb" {
  statement {
    effect = "Allow"
    actions = [
      "dynamodb:Query",
      "dynamodb:GetItem",
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
    ]
    resources = "arn:aws:dynamodb:eu-west-1:012345678901:table/mytable"
  }
}
```

```
data "aws_iam_policy_document" "dynamodb" {
  statement {
    effect = "Allow"
    actions = ["dynamodb:*"]
    resources = ["*"]
  }
}
```

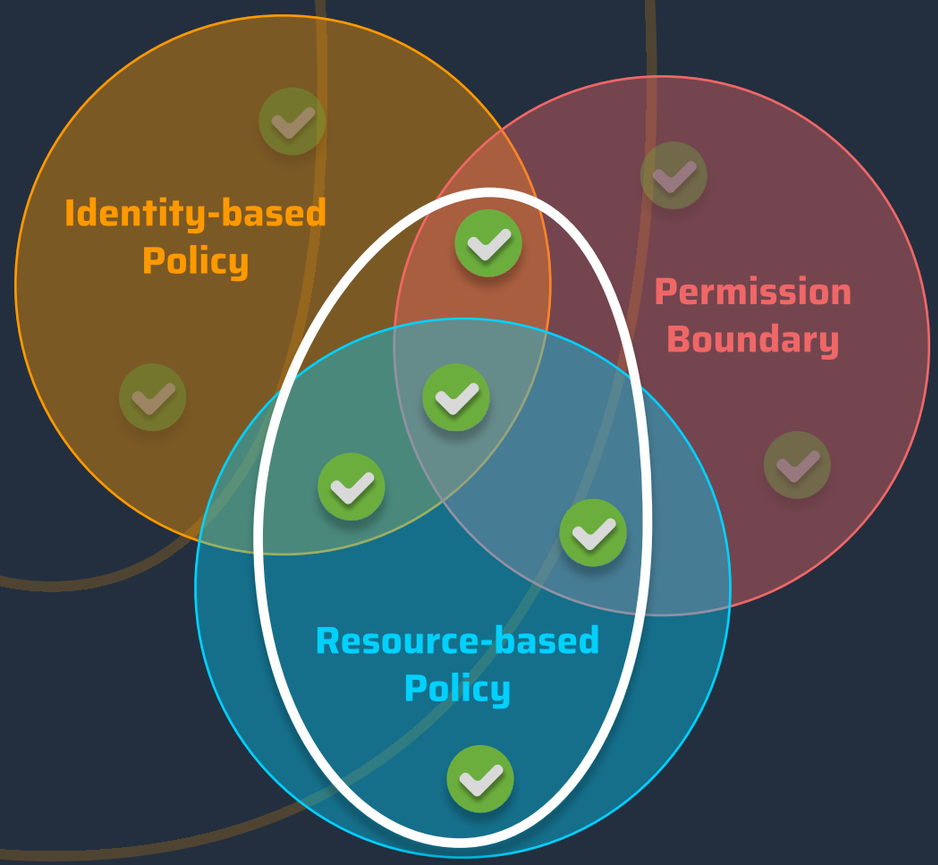
Even Better: directly referencing your Terraform resources!

AWSFUNDAMENTALS.COM

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.

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.

AWSFUNDAMENTALS.COM

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.

AWS CLI 

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 default 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**.

Great free community tools are:

- **AWSume** - awsu.me: a terminal-based solution.
- **Leapp** - leapp.cloud: a GUI-based solution that also supports other cloud providers like Microsoft Azure.

Both are easy to set up and work with. Don’t miss out on one of them to avoid frustrations.

Interested in Learning AWS for the real world?

Check out our **newsletter**, **fundamentals book**, and **socials** 🪄

tpschmidt

@tpschmidt_

alessandro-volpicella

@sandro_vol

AWSFUNDAMENTALS.COM