**IAM:**

Identity and access management, also known as IAM controls access to all of the AWS API endpoints. It controls the authentication & authentication.

**Root User:**

* Root user is created automatically for an account and it has unrestricted access and full control of the account.
* For an isolated AWS Account there isn't a real way to restrict the access for a root user (Exception AWS Organization).
* The account root user is responsible for the initial setup of IAM inside an AWS account.

**IAM User's sign in link:**

* This is the Link used by IAM Users to login to the AWS Account.
* **Looks like**: https://<Account\_id>.signin.aws.amazon.com/console
* **Example:** <https://323131968025.signin.aws.amazon.com/console>
* We can also **define an alias for the account id**. For example if the alias is "subhro2606", the sign in link would look like <https://subhro2606.signin.aws.amazon.com/console>

**Password Policy:** The password policy governs minimum password length and other limitations and requirements of the passwords and initially the root user is responsible for setting it.

**Architectural Components of IAM:**

* IAM users
* IAM groups
* IAM roles
* IAM policies

IAM **controls** **access** to AWS services via **policies** that can be attached **users, groups, and roles**. **Users** are given **long-term credentials** to access AWS resources (**username and password or access keys**).

**Roles** allow for **short-term access** to resources when assumed, using **temporary access credentials**.

**ARN (Amazon Resource Name):**

ARN can uniquely identify any resource in any region in any account.

**ARNs always begin with:**

arn : partition : service : region : account-id:

* **partition** = aws or aws-cn (for China)
* **service** = the AWS service: s3, ec2, rds, dynamodb
* **region** = region code: us-east-1, ap-southeast-2

**And, depending on service, finish with:**

* resource
* resourcetype/resource
* resourcetype/resource/qualifier
* resourcetype/resource:qualifier
* resourcetype:resource
* resourcetype:resource:qualifier

**Example ARNs:**

* arn:aws:iam::123456789012:user/roffle
* arn:aws:s3:::myamazingcatpics/truffs.jpeg
* arn:aws:dynamodb:us-east-1:123456789012:table/ratemycats

**In some cases, wildcards are supported:**

* arn:aws:ec2:us-east-1: 123456789012:instance/\*
* Fields with :: **omit the value**, and \* is a **wildcard match**.

**IAM Policy:**

An IAM policy (JSON policy document) is known as an identity policy when attached to an identity or a resource policy when attached to a resource. They have no effect until they are attached to something.

A policy document is a list of statements:

{

"Version": "2012-10-17",

"Statement": [{...}, {...}, {...}]

}

Each statement matches a request to AWS. Requests are matched based on their Action (or actions), which are the API calls or operations being attempted and the Resource (or resources) the request is against. A given statement results in an Allow or Deny for the request.

{

"Sid": "SpecificTable",

"Effect": "Allow",

"Action": ["dynamodb:BatchGet\*", "dynamodb:describeStream", "dynamodb:DescribeTable", "dynamodb:Get\*", "dynamodb:Query", "dynamodb:Scan", "dynamodb:BatchWrite\*", "dynamodb:CreateTable", "dynamodb:Delete\*", "dynamodb:Update\*", "dynamodb:PutItem"],

"Resource": "arn: aws:dynamodb:\*:\*:table/CatPics"

}

**Managed Policies:**

Managed policies allow the **same policy to impact many identities**. Now managed policies are defined inside the policy section of the IAM console.

There are two types:

* **AWS Managed Policies:**

These are managed by AWS.

* + (+) They have **little admin overhead**.
  + (-) **Lack flexibility** because you're relying on AWS to define these policies for you so they're not going to fit your exact business needs.
  + **Examples** of AWS managed policies are Amazon chime read only, Access Amazon API Gateway Administrator, Administrator Access, which gives admin functionality of to an IAM user, and there's many more and then you've also got customer managed policies.
* **Customer Managed Policies:**

These are created and managed by the customer.

* + (-) They have **more admin overhead**.
  + (+) **Completely flexible**.

**Inline Policies:**

Policy documents could be **directly added to identities so users, groups, and roles** and these are known as inline policies.

Inline policies **allow exceptions to be applied to identities**.

**Exam Tips:**

* If a request isn't explicitly allowed, it's **implicitly (default) denied**.
* If a request is **explicitly denied**, it **overrides everything else**.
* If a request is **explicitly allowed**, it's **allowed unless denied by an explicit deny**.
* Remember: **DENY -> ALLOW -> DENY**
* Only **attached policies have any impact.**
* When **evaluating policies**, all **applicable policies are merged**: All identity (user, group, role) and any resource policies
* **Managed policies** allow the same policy to impact many identities.
* **Inline policies** allow exceptions to be applied to identities.
* **AWS-managed policies** are low overhead but lack flexibility.
* **Customer-managed policies** are flexible but require administration.
* **Inline and managed policies** can apply to users, groups, and roles.

**IAM Users:**

IAM users are a type of IAM identity suitable for **long-term access for a known entity** (human, service, application).

**Authentication:**

Principals authenticate to IAM users either with a username and password or using access keys.

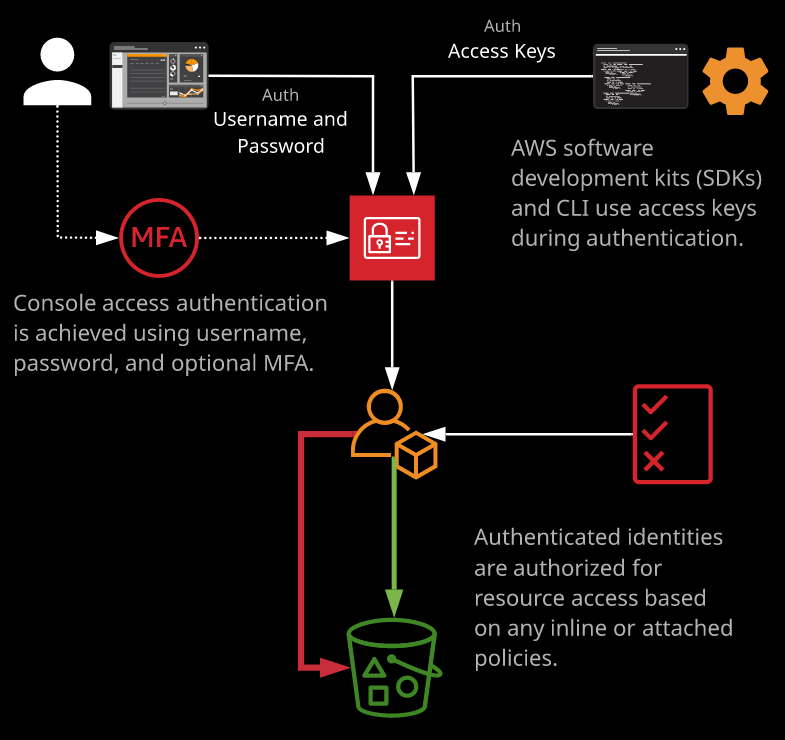
So, Principals actually **login** to as an IAM Users.

* **Username & Password:** Generally, username & password (optionally MFA) is used for authentication when Principal is logging into the console.
* **Access keys:** While using the Command Line or APIs directly, principals can use the access keys.

**Authorization:**

Once the user is authenticated, the **applicable IAM policies** (applicable directly to the user or applicable to the group the user is part of) for the IAM User are used for authorization, to allow/deny access an aws resource.

If no policies are defined the user can do nothing.



**Real Identity:**

Anything that has an ARN and can be referenced from elsewhere within AWS called a Real Identity. For example- IAM Users, Resource Polices.

**Non-Real Identity:**

Doesn't have an ARN and can’t be referenced. For example-Groups are not real identities.

**Exam Facts and Figures:**

* If you see any exam questions, where there is **"login"** mentioned then it has to be an **IAM user**. Since groups and roles can't log in or be logged in to.
* Use **managed policies** for baseline permissions, permissions that you want to assign to many different identities, and then best practice is to use **inline policies** for any exceptions to that.
* Hard limit of **5000 IAM users per account** — this is important it can impact architecture.
* **10 group memberships** per IAM user.
* Default maximum of **10 managed policies per user or per group.**
* No inline limit, but you **cannot exceed 2048 characters for all inline policies** on an IAM user
* 1 MFA per user
* 2 access keys per user

**IAM Group:**

An IAM group is a **collection of IAM users**.

Groups allow **easier administration** over sets of IAM users.

**Inline** and **managed policies** can be **applied to groups** that **flow on to members of that group**.

Groups are **not a true identity** — they **cannot be the principal in a policy**, so they can't be used in resource policies.

**Exam Facts and Figures:**

* Groups are an admin feature to **group IAM users**.
* **Groups** can **contain many IAM users**, and **users can be in many groups**.
* **IAM In line policies** can be **added to IAM groups** — and these **flow on to IAM users who are members**.
* **Managed IAM policies** can be attached and **flow on to IAM users who are members**.
* Groups are **not "true" identities**, and they **can't be referenced from resource policies**.
* Groups have **no credentials**.
* Note that a group is not truly an "identity" in IAM because it cannot be identified as a Principal in a permission policy. It is simply a way to attach policies to multiple users at one time.
* **Hard limit of 300 groups** per AWS account.

**AWS Access Keys:**

Access keys are a pair of values used by applications, **SDKs, or the AWS command line** to authenticate to AWS. They are **NOT used for logging into the console**

Access keys consist of two parts: the access key ID and secret access key.

**Access key ID:** The access key ID is the public part of the key and is stored by AWS once generated.

Access Key ID: AKIAIOSFODNN7EXAMPLE

**Secret Access Key:** The secret access key is the sensitive and private part of the access key, available only once when the access key is initially generated. It's stored only by the owner and should never be revealed.

Secret Access Key: w3a1rXUtnFEMI/K7MDENG/bPKRfiCYEXAMPLEKEY

An **IAM user is the only identity that uses access keys**. They are allowed two sets. They can be created, deleted, enabled, and disabled.

AWS allows a **maximum of two access keys per AWS user**. It is not about whether the status is active (enabled) or inactive(disabled). There can be a maximum of two access keys active(enabled)/inactive(disabled).

They **can't be used to log in to the console**, and they **don't expire**.

If anyone finds a set of access keys, they have access to the permissions of the IAM user to which they belong.

**Operation on the access Keys:**

* Now, in terms of what operations you can do with an access key well currently have **created it** and it's **active**.
* You can **disable an access key** and the AWS terminology for that is to make the key **inactive**.
* You're able to make an **inactive key active again**.
* You can also **delete an access key** when you delete it. It's removed entirely from the system, and you can't use it anymore to interact with AWS.
* Now AWS do recommend that these are **rotated regularly.**
* Access keys don't expire when you create them. They work until you make them inactive or you delete them.
* Now AWS do recommend that these are **rotated regularly**. So, if you have any uses for an access key than what you should do, really is to **keep the user tied down to one of them**, and then periodically you can create another access key. (Read: <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html#rotating_access_keys_console>)

**Authentication & Authorization:**

* **Access keys are checked every time they used**. Whenever an **API call is made from the command line or using the SDK**, you're using a pair of access keys, and every single time that that occurs, you're **authenticated**.
* Using those keys and then based on which **IAM user those keys belong to you get a set of policies** and then you're **authorized based on their set of policies.**
* So, if you make **access keys inactive** than immediately, every **usage** of that access key pair is **denied**.
* If an access key pair is **active**, so you're able to **use it** and you **update the policies that are associated with the identities of the user**, that Access Key belongs to. Then **the permissions** that there were on the access keys, is immediately adjusted.
* So that's important to keep in mind **every single time that these access keys are used** they checked for validity and then you are **authorized based on the policies that the user has at the time that they're used**.

**IAM roles**:

IAM Roles are assumed by another identity allowed in the trust policy — IAM user, AWS service, another AWS account, web identity, or even an anonymous identity.

When a role is assumed, the Security Token Service (STS) generates a time-limited set of access keys (temporary security credentials). These access keys have the permissions defined in the permissions policy. IAM roles have no long-term credentials (access keys or username and password).

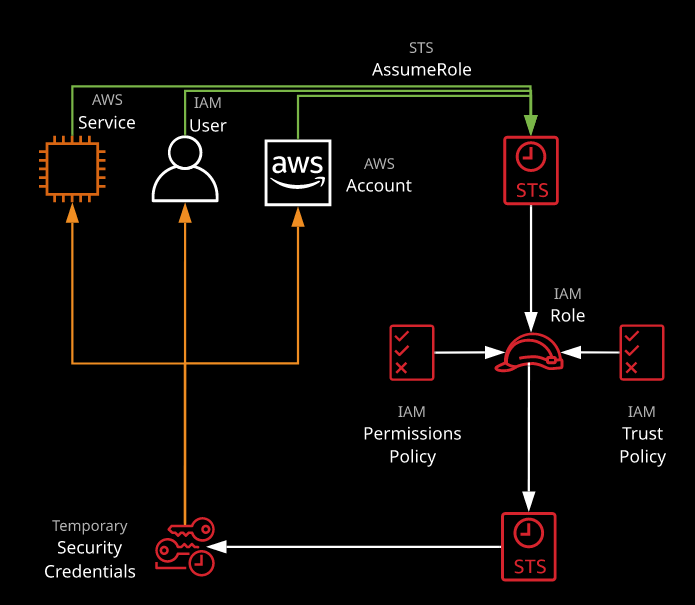
**Trust Policy:**

Trust policies define list of entities or identities or services that can assume a role.

**Permissions Policy:**

The permissions policy is an IAM policy that gives that role permissions on things. It's the same as an IAM policy on a user or an IAM policy on a group. It basically defines the actions, the entities defined in the trust policies can perform.

**How Role works:**

* Roles are used to give AWS services permissions on your behalf.
* When somebody assumes a role. They make an assume role API call which ask AWS let that entity assume a role.
* If they're on the trust policy so if this service or this user or somebody in this remote AWS account is allowed to assume the role, then they actually assume that role, and when they assume the role, a service called STS, which is the security token service that gets involved.
* STS generates what's known as a temporary security credentials. These are like access keys that we talked about in a previous lesson, but they are time limited. So, if STS determines that somebody can assume a role it generates temporary security credentials and those temporary security credentials are basically time limited access keys that get the permissions based on the permissions policy of the role.
* This is essentially just allowing any identity to temporarily make use of permissions that a role has.

**Exam Facts and Figures:**

• IAM roles have **no long-term credentials.**

• They are sts:AssumeRole by another identity:

* IAM users
* AWS services
* External accounts
* Web identities

• **Temporary security credentials** are generated by **STS.**

• **Trust policy** controls which identities can assume the role.

• **Permissions policy** defines the permissions provided.

• Temporary credentials expire.

• Example scenarios:

* Company merger
* "Break-glass"-style extra access
* Cross-account access
* AWS service access
* Web identity federation

**AWS Organizations:**

AWS Organizations is a service for managing multiple accounts within a single business. Rather than managing many accounts, with many isolated sets of logins and individual bills, Organizations allows consolidation.

All accounts within an AWS Organization can consolidate bills into a single account — one bill covering all business usage.

Organizations can share bulk discounts and even easily manage accounts and permissions and limit account usage using service control policies.

**Organization is global service**, so you don't have to pick a particular region but what you need to do is create an organization.

**Modes:**

Now an organization can operate in one of two modes.

* **Standard Mode:** Also called **all features mode,** includes all features, which includes consolidated billing along with extra functionalities that help you administer lots of different accounts.
* **Consolidated Billing Mode:** Consolidated billing allows is individual AWS accounts to be combined to generate a single bill.

We can use consolidated billing, but recommendation is to use utilize the **Standard Mode or All Features Mode**.

Once organization is configured, so any other accounts will automatically consolidate their billing into this master account.

**Standard Account:** Every AWS account when you first create it is a standard account.

**Master Account:** When you create an organization, the current standard account gets converted to being the master account for this organization. So, the first account inside AWS Organizations becomes the master account within that organization.

There can only ever be **one master account**, and the master account **can't be restricted** in any way.

**Use Case for the Master Account:**

* **Consolidated Billing:** So best practice is that generally this master account is only used for the consolidated billing part of the organization.
* **Centralized Logging:** It can also be used for things like centralized logging. So, all of the different accounts in the organization can push their logs into this master account.
* **IAM Users:** It can also be used for IAM users for your entire organization. So, you might have all of your logins occurring to this master account and then roles are used to switch into any other member accounts.

**Member Account:**

Any accounts that are in the AWS Organization except the master account, are member accounts.

**Financial Benefits of Using AWS Organizations**:

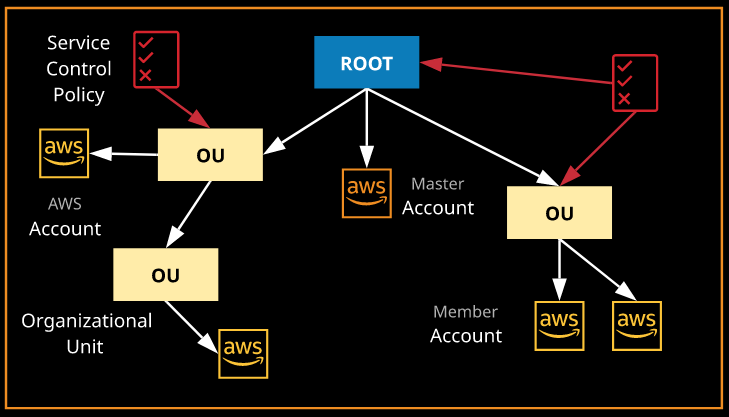
By utilizing AWS organizations and consolidated billing, any reserved instances purchased in these individual accounts are combined into this master account, so as an organization, the benefits from having a **bigger pool of resources**. Since **anything that is purchased is spread over all of the accounts inside the organization**, organizations from a financial perspective benefit, from this **reduced admin overhead** because you only have the single bill, but also you get benefits from the increased usage as well as being able to share any reserved purchases.

**Architecture:**

Organizations are operating in the **all features mode** benefit from what are called service control policies. So, this is a structure of an AWS organization.

**Root Node:** Organizations operate with an inverse tree architecture and root node is the top of the organization. Everything flows down from the root.

**Organization Units:** There is only one root note, and under root node there can be AWS accounts, or you can have organizational units and they can contain either accounts or they can contain other organizational units.

So, you can build up a fairly **complex tree style architecture**.

**Service control policies (SCPs):** Service control policies restrict what accounts can and can't do, but the master account cannot be restricted in any way and so it really is important to consider your account architecture. These SCP(s) can be applied to individual accounts, to organizational units, or to the root node and they affect any AWS account that is at that level or below it.

So, if you apply a service control policy to the root node, then it applies to any AWS accounts in that root node or any organizational units in that root node and any organization units in the organizational unit and any accounts and so on.

So whatever level you apply these policies they impact that level and anything below it.

If you apply a service control policy to a specific account, it only effects that account.

If you apply it to an OU it effects that OU so any accounts in the OU and any OUs in the OU and any accounts in the OU and so on.

Now there's one exception to this, and that's the master account.

Master accounts cannot be restricted, and that's why it's especially important to decide in advance if you want the **master account to be dedicated to just being a master account** because you can't restrict it.

**Use Case for SCP:**

* **Restrict AWS accounts** to only be able to operate in specific AWS regions,
* **Restrict what products** can be accessed from that account.

The key architectural point is that **if you do use the master account to host any resources**, you won't be able to restrict what gets used inside that master account.

**Limit:**

We can add additional accounts of this organization bucket but there are relevant account limits.

The base limit is two accounts inside an organization, however the same can be increased by logging a service limit related ticket to increase the account limits.

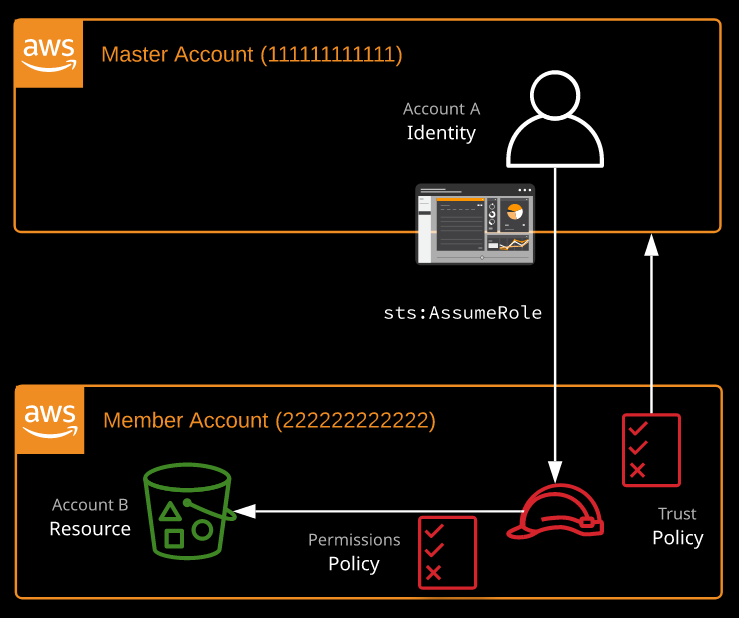
**Role Switching:**

Role switching is **a method of accessing one account from another using only one set of credentials.**

It is used both within **AWS Organizations** and **between two unconnected accounts**.

1. A role (Generally this is "**OrganizationAccountAccessRole**") in Account B trusts Account A.

2. An identity in Account A can assume the role in Account B.

3. Using that role, it can operate inside Account B.

**OrganizationAccountAccessRole:**

When we add account inside an organization, you can either **invite an account**—an existing AWS account—or you can **create a brand-new account** within the organization.

* **OrganizationAccountAccessRole Automatically Created:** "**OrganizationAccountAccessRole**" is automatically created for **newly created AWS Account**, which **allows us to role switch into that account** from this master account.
* **OrganizationAccountAccessRole not Automatically created:** In case we are **adding additional accounts** to an organization so **not creating new accounts** within the organization, so for **inviting accounts**, this role would **NOT be automatically created**.

If we are inviting accounts, we'll need to get the account root user or an account IAM user inside that account to create that IAM Role and from best practice perspective call the role "OrganizationAccountAccessRole" and make sure that it trusts the master account using the trust policy and then, from that point onward, we can treat it exactly the same as we would as if you created the account inside the organization.

**Use Case:**

**Single account with all of the identities:** We can have a single account with all of the identities within it. And then these identities can be allowed to potentially role switch into hundreds or more member accounts.

**Active Directory for Corporate Logins:** Instead of using IAM Identities like IAM User credentials for Corporate login can set up active directory to login to the master account then the user can role switch to other accounts

**Service Control Policies:**

Service control policy essentially **provides a boundary a boundary** that anyone inside the member account to which it's applied can't breach.

So, in the **member account**, even if we are logged in as the **account root user**, the service control policy would still **restrict** what we are able to do inside this account.

Remember, service control policies **don't affect the master account** in any way but for any other accounts or any other member account, they apply to every single identity in that account, including the account root user.

**Exam Tip:**

* The only way to **restrict the ability of a root user** in an account to do anything, is by using **service control policies.** And that can only be used if we are **inside an organization** and only then if we've enabled the use of service control policies by enabling it at the root and then applying it either to the root, the OU, or the account itself.
* However, the **master account cannot be restricted** in any way.