Identity and Access Management (IAM)

IAM is used to define user access permissions within AWS.

Gives you centralized control of an Aws account.

Is a Global service – no concept of region for this service i.e. all users, groups, policies etc.. are available in all the regions.

Can be used to give temporary access.

IAM terms

- Resources – The user, group, role, policy, and identity provider objects that are stored in IAM. You can add, edit, and remove resources from IAM.

- Identities – The IAM resource objects that are used to identify and group. Attach a policy to an IAM identity. These include users, groups, and roles.

- Entities – The IAM resource objects that AWS uses for authentication and include users and roles. Roles can be assumed by IAM users and roles in your or another account. They can also be assumed by users federated through a web identity or SAML.

- Principals – A person or application that uses the AWS account root user, an IAM user, or an IAM role to sign in and make requests to AWS.

Features of IAM:

- Central control of users and security credentials.
- Central control of user access
- Shared AWS resources i.e. Users can share data for collaborative projects.
- Permissions based on organizational groups.
- Single AWS Bill i.e. your organization's AWS account gets a single AWS bill for all your user's AWS activity.
- Networking Controls – You can help make sure that users can access AWS resources only from within the organization's corporate network using SSL, etc.

IAM Users

- Users are the individual accounts created in AWS.
- By default, new users don't have access to any AWS services.
- Always set up MFA for your root account.
- Newly created IAM users have no password and no access key.

- If the user needs to administer your AWS resources using the AWS Management Console, you can create a password for the user. If the user needs to interact with AWS programmatically (using the command line interface (CLI), the AWS SDK, or service-specific APIs), you can create an access key for that user.
- Each user is associated with one and only one AWS account.
- An IAM user doesn't necessarily have to represent an actual person. An IAM user is really just an identity with associated permission.
- "You're creating an app that runs on a mobile phone and that makes requests to AWS." – In this scenario, don't create an IAM user and distribute the user's access key with the app. Instead, use an identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google to authenticate users, and then use that identity to get temporary security credentials.

To create an IAM Users (via console)

1. Sign into the AWS Management Console and open the IAM console at https://console.amazonaws.com/iam/. API Version 2010-05-08 44 AWS Identity and Access Management Using IAM Overview
2. In the navigation pane, click Users and then click Create New Users.
3. Enter the usernames for the users you want to create. You can create up to five users at one time.
4. If you want to generate an access key ID and secret access key for new users, select Generate an access key for each user. Users must have keys if they need to work with the AWS CLI or with the AWS SDKs or APIs. Click Create.
5. A page appears that enables you to download the access key IDs for the new user or users.
6. To save the access keys for the new user or users, click Download Credentials. This lets you save the access key IDs and secret access keys to a CSV file on your computer.

Via CLI or API:

Create a user : aws iam create-user

API: CreateUser

Give Password to user: aws iam create-login-profile

API: CreateLoginProfile

Change user's password: aws iam update-login-profile

Create AccessKey for user: aws iam create-access-key

API: CreateAccessKey

Attach Policy to user: aws iam attach-user-policy

Add users to one or more group: aws iam add-user-to-group

List all users in account: aws iam list-users

API: ListUsers

List users in specific Group: aws iam get-group

List all the groups that user is part of: aws iam list-groups-for-user

Rename users: aws iam update-user

Delete user’s key: aws iam delete-access-key

Delete certificate assigned to user: aws iam delete-signing-certificate

Delete user’s password: aws iam delete-login-profile

Delete policy from user: aws iam detach-user-policy

Delete user: aws iam delete-user

IAM Groups

- A collection of IAM users.
- Simply the assignment of permissions.
- Examples like Operations, Admin, Tester, Development, Finance groups.
- A user can belong to multiple group (10 max)
- A group can contain many users, and a user can belong to multiple groups.
- Groups can't be nested; they can contain only users, not other groups.
- There's no default group that automatically includes all users in the AWS account. If you want to have a group like that, you need to create it and assign each new user to it
- Group deletion requires detaching users and managed policies and delete any inline policies.

To add or remove users in group (via console)

1. Sign into the AWS Management Console and open the IAM console at https://console.amazonaws.com/iam/.

2. In the navigation pane, click Groups, and then click the name of the group.

3. Open the Users section.

4. To add users to the group, click Add Users to Group. To remove users from the group, click Remove Users from Group.

5. Select the users you want to add to the group or remove from the group, and then click Add Users or Remove Users.

To attach a policy to a group (via console)

1. Sign into the AWS Management Console and open the IAM console at https://console.amazonaws.com/iam/.

2. In the navigation pane, select Policies.

3. In the list of policies, select the check box next to the name of the policy to attach. You can use the Filter menu and the Search box to filter the list of policies.

4. Click Policy Actions, then click Attach.

5. Click All Types in the Filter menu, then click Groups.

6. Select the check box next to the name of the group to attach the policy to, then click Attach Policy.

Note: We saw how to create group, add policy in our session. Let's see the commands here to be used while working with AWS CLI.

Create a group: aws iam create-group

API: CreateGroup

Attach a Policy to the Group: aws iam attach-group-policy

API: AttachGroupPolicy

Add a user to the Group: aws iam add-user-to-group

API: AddUserToGroup

Remove a user from Group: aws iam remove-user-from-group

API: RemoveUserFromGroup

List all groups: aws iam list-groups

List users in group: aws iam get-group

Rename a group: aws iam update-group

Delete all inline poliices embedded in group: aws iam list-group-policies

Detach a policy: aws iam detach-group-policy

Delete the Group: aws iam delete-group

Common Query on MFA Device Lost or Stop Working?

If an MFA device stops working, is lost, or is destroyed, and you can't sign in to the AWS portal or the AWS Management Console, then you need to deactivate the device. AWS can help you deactivate the device. The way you get help depends on whether an MFA device is assigned to the root account or to a user under an AWS account.

To get help for an MFA device associated with an AWS root account

1. Go to the AWS Contact Us page for help with disabling AWS MFA so that you can temporarily access secure pages on the AWS website and the AWS Management Console using just your user name and password.

2. Change your AWS password in case an attacker has stolen the authentication device and might also have your current password.
3. If you are using a hardware MFA device, contact the third-party provider Gemalto using their website for help fixing or replacing the device. If the device is a virtual MFA device, delete the old MFA account entry on the device before creating a new one.
4. After you have the new physical MFA device or you have completed deleting the old entry from the mobile device, return to the AWS website and activate the MFA device to re-enable AWS MFA for your AWS account

To get help for an MFA device associated with an IAM user

Contact the system administrator or other person who gave you the username and password for the IAM user. They will need to deactivate the MFA device.

IAM Role:

A role is essentially a set of permissions that grant access to actions and resources in AWS. These permissions are attached to role, not to an IAM user or group. Roles can be used by an IAM user in the same AWS account as role or a different account, an AWS service such as EC2, or an identity provider.

IAM role does not have passwords or keys but whoever assumes the role is provided temporary credentials.

IAM best practice is to use roles for delegation instead of sharing credentials.

Another best practice is to use IAM roles for applications running in EC2 instances.

Cross Account Access: Granting access to the resources in one account to a trusted principal in different account is often referred to as cross-account access. Roles are primary way to grant cross-account access.

Let's see a simple example of creating an AWS IAM Role which allows API Gateway to perform READ and WRITE operations on Dynamo DB.

1. Login to the console, go to IAM Dashboard and click on Roles - > Create Role.
2. Select API Gateway from the list of options. You will be presented with the default use case selection of "API Gateway" allowing API to push logs to CloudWatch.
3. Click "Next: Permissions"
4. Attach the Permission Policies named as " AmazonAPIGatewayPushToCloudWatchLogs"
5. Click "Next: Tags" to proceed.
6. Give the desired role name. In Role Description mention as "Allows API Gateway to push logs to CloudWatch Logs and full access to DynamoDB."
7. Click "Create Role"

The new role has been created and you should be presented with the roles dashboard. We now need to add dynamodb policy to the role. To do this, find the new role and click on the role name to be taken to the role summary view. Click the Attach Policies button and find the **AmazonDynamoDBFullAccess** policy. Select the policy and click “Attach Policy”. This new role can be used within API Gateway to access DynamoDB.

Refer to the recording of session on how to create IAM Roles for different scenarios.

What is an IAM Policy?

An IAM Policy is an object in AWS that, when associated with identity or resource, defines their permissions. User manage access in AWS by creating policies and attaching them to identities (users, groups or roles) or AWS resources. When you create a permissions policy to restrict access to a resource, you can choose an Identity-Based Policy or a resource-based policy.

Most policies are stored in AWS as **JSON** documents.

Among the two types of policies, **Identity-Based Policy** include AWS managed policies, customer managed policies and inline policies.

AWS Managed Policies are created and managed by AWS while **Customer Managed Policies** are managed by users itself in their respective AWS account.

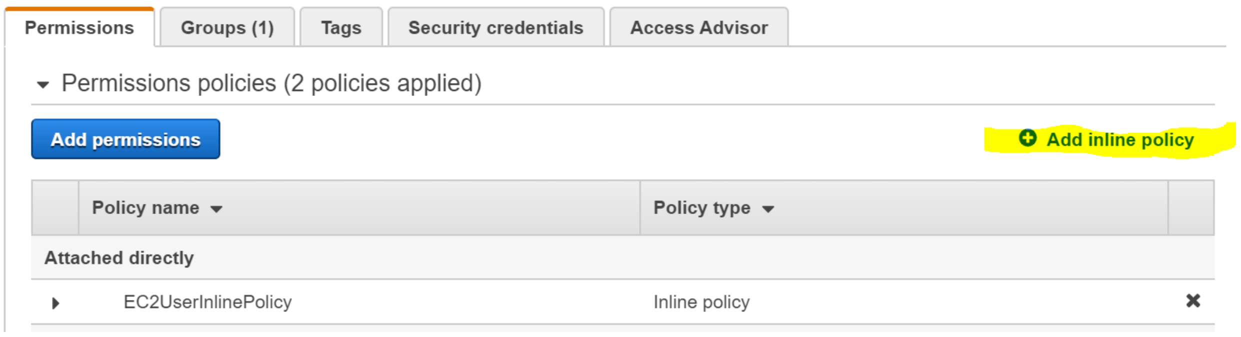
Managed Policies for common use cases based on job function (e.g. AmazonDynamoDBFullAccess, AWSCodeCommitPowerUser, AmazonEC2ReadOnlyAccess, etc.). These AWS-provided policies allow you to assign appropriate permissions to your users, groups, and roles without having to write the policy yourself. A single Managed Policy can be attached to multiple users, groups, or roles within the same AWS account and across different accounts. You cannot change the permissions defined in the AWS Managed Policy.

Customer Managed Policy is a standalone policy that you create and administer inside your own AWS account. You can attach this policy to multiple users, groups, and roles; but only within your own account. In order to create a Customer Managed Policy, you can copy an existing AWS Managed Policy and customize it to fit the requirements of your organization. Recommended for use cases where the existing AWS Managed Policies don't meet the needs of your environment.

Inline Policy is an IAM policy which is embedded within the identity. Don’t forget that there is a strict 1:1 relationship between the entity and policy. When you delete the user, group, or role in which the Inline policy is embedded, the policy will also be deleted. Inline Policies are useful when you want to be sure that the permissions in a policy are not inadvertently assigned to any other user, group, or role than the one for which they're intended (i.e. you are creating a policy that must only ever be attached to a single user, group, or role).

In general, you should avoid using inline policy in AWS as they are difficult to manage and you must go to the individual entities, such as groups, to make any required changes. Following are the steps to create an Inline Policy in AWS.

1. Select the user, group or role entry in Navigation Page of IAM at console.
2. Open the entity you want to work with by clicking its entry in the ObjectType Page.
3. Go to the Permissions Tab.
4. Select Add Inline Policy to create one as highlighted in screenshot:

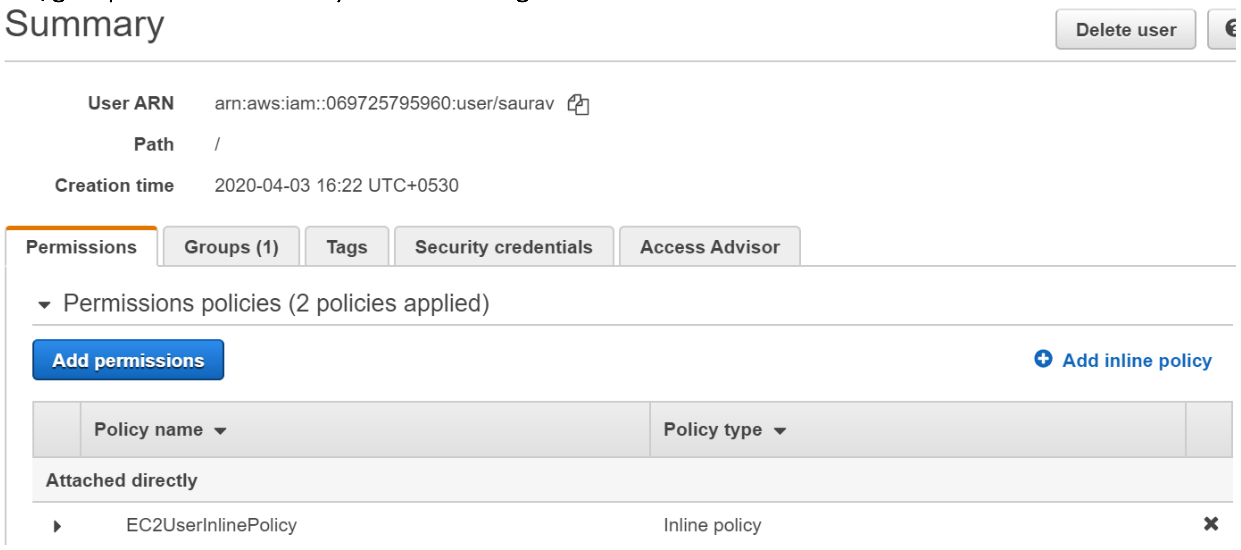

Permissions | Groups (1) | Tags | Security credentials | Access Advisor

Permissions policies (2 policies applied)

Add permissions | Add inline policy

	Policy name	Policy type	
Attached directly			
	EC2UserInlinePolicy	Inline policy	✖

5. Now you can create the Policy as per your choice, this policy will be added to individual user/group or role for which you are creating.


Summary

Delete user

User ARN arn:aws:iam::069725795960:user/saurav

Path /

Creation time 2020-04-03 16:22 UTC+0530

Permissions | Groups (1) | Tags | Security credentials | Access Advisor

Permissions policies (2 policies applied)

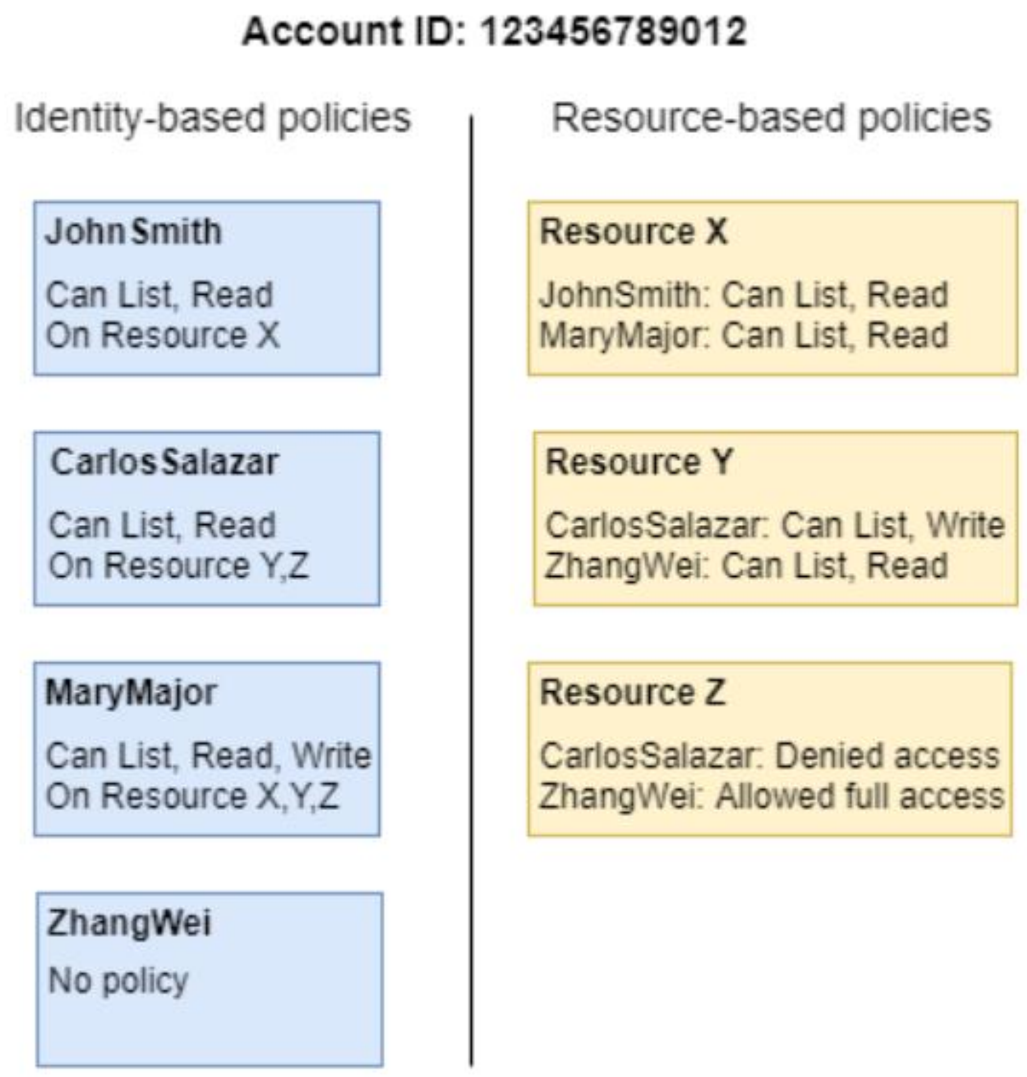
Add permissions | Add inline policy

	Policy name	Policy type	
Attached directly			
	EC2UserInlinePolicy	Inline policy	✖

6. While you are creating the policy, you might encounter following error: "Your policy character exceeds the non-whitespace character limit of 2,048." It is because objects in AWS IAM and AWS STS (Security Token Service) has size limitation which basically limits you to how you name an object, the number of objects you create, and number of characters you can pass on object. Note: Policy name for inline policies must be unique to the user, group or role they are embedded in. The names can contain any Basic Latin (ASCII) characters minus the following reserved characters: backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space.
Above error is as we have exceeded the character limit for our policy i.e. 2048 chars. Hence need to change the policy parameters and define again. Please see below in Limitations of IAM section for actual limit set by Amazon.

For more info on limits, visit or see the section Limitation of IAM Entities in this document.
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_iam-limits.html

Illustration of Policy:



Source: AWS

The administrator of the 123456789012 account attached *identity-based policies* to the JohnSmith, CarlosSalazar, and MaryMajor users. Some of the actions in these policies can be performed on specific resources. For example, the user JohnSmith can perform some actions on Resource X. This is a *resource-level permission* in an identity-based policy. The administrator also added *resource-based policies* to Resource X, Resource Y, and Resource Z. Resource-based policies allow you to specify who can access that resource. For example, the resource-based policy on Resource X allows the JohnSmith and MaryMajor users list and read access to the resource.

The 123456789012 account example allows the following users to perform the listed actions:

- JohnSmith – John can perform list and read actions on Resource X. He is granted this permission by the identity-based policy on his user and the resource-based policy on Resource X.
- CarlosSalazar – Carlos can perform list, read, and write actions on Resource Y, but is denied access to Resource Z. The identity-based policy on Carlos allows him to perform list and read actions on Resource Y. The Resource Y resource-based policy also allows him write permissions. However, although his identity-based policy allows him access to Resource Z, the Resource Z resource-based policy denies that access. An explicit Deny overrides an Allow and his access to Resource Z is denied.
- MaryMajor – Mary can perform list, read, and write operations on Resource X, Resource Y, and Resource Z. Her identity-based policy allows her more actions on more resources than the resource-based policies, but none of them deny access.
- ZhangWei – Zhang has full access to Resource Z. Zhang has no identity-based policies, but the Resource Z resource-based policy allows him full access to the resource. Zhang can also perform list and read actions on Resource Y.

Identity-based policies and resource-based policies are both permissions policies and are evaluated together. For a request to which only permissions policies apply, AWS first checks all policies for a Deny. If one exists, then the request is denied. Then AWS checks for each Allow. If at least one policy statement allows the action in the request, the request is allowed. It doesn't matter whether the Allow is in the identity-based policy or the resource-based policy.

IAM Policy Structure

There are two ways you can create IAM policies from IAM web console. Visual Editor and a character-based JSON policy editor. However, we focus on the JSON policy which can give fine-grained customized control over the resources. Once the policy is created it can be attached to user, group or role. The JSON policy document consists of the following elements:

Effect –Allow or Deny access to the resource is decided by Effect (Allow/Deny)

Action — A set of service-specific parameters (like “iam: CreateUser”).

Resource — Resource names (like “arn:aws:s3:::conf-* “)

Condition (Optional) — Grant conditions (like “aws: RequestedRegion”: “ap-south-1”)

IAM Policy Evaluation

- By default, all requests are denied except for root.
- An explicit allow overrides this default.
- An explicit deny overrides any allows.

Let’s see some useful IAM Policies

User Related: Creation of delegated Account Operator

Policy 1: Delegate read permissions for IAM components.

```
{
“Version”: “2012–10–17”,
“Statement”:
 {“Effect”: “Allow”,
“Action”: [“iam:Get*”,“iam:List*”],
“Resource”: “*”
}
}
```

Policy 2: Delegate Account Operator who can create the user. Remember, that he is not yet been given right to add the user to group yet (see policy 3 for the same).

```
{
"Version":"2012-10-17",
"Statement": [{
"Effect":"Allow",
"Action":["iam:ListUsers","iam:ListGroups","iam:GetGroup","iam:CreateUser","iam:UpdateUser","iam:Delete
User","iam:CreateVirtualMFADevice","iam:EnableMFADevice","iam:DeactivateMFADevice","iam:DeleteVirtua
lMFADevice","iam:CreateLoginProfile","iam:UpdateLoginProfile","iam:DeleteLoginProfile"],
"Resource":"*"
},
{
"Effect":"Allow",
"Action": ["iam:GetAccount*","iam:ListAccount*"],
"Resource":"*"
}
]
}
```

Policy 3: Delegates Account Operator to add the user to groups *Developers* and *Operators* only. (you may have to combine above two policies to get all the relevant permissions)

```
{
```

```
"Version": "2012–10–17",
"Statement":
{
"Effect": "Allow",
"Action":["iam:AddUserToGroup","iam:RemoveUserFromGroup","iam:GetGroup"],
"Resource":["arn:aws:iam::609103258633:group/Developers","arn:aws:iam::609103258633:group/Operators
"]
}
}
```

Region Related Policy

Policy 4: User is allowed to access EC2 resources from ap-south-1 (Mumbai) only.

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [ "ec2:*" ],
"Resource": "*",
"Condition": {
"StringEquals": {
"aws:RequestedRegion" : "ap-south-1"
}
}
}
]
}
```

Policy 5: EC2 service is available only when the user access the AWS web console form an iP address 192.168.0.1

```
{
"Version" : "2012-10-17",
"Statement" : [
"Effect" : "Allow",
"Action" : "EC2:*",
"Resource" : "*",
"Condition" : {
"ipAddress" : {
"awsSourceIP": [ "192.168.0.1"]
}
}
}
}
```

S3 Related:

Policy 6: A power user is prevented from accessing /deleting any bucket starting with name "conf-". Assuming the user has full permission for the bucket before applying the policy.

```
{
"Version" : "2012-10-17",
"Statement" : [
{
"Sid" : "VisualEditor1",
"Effect" : "Deny",
"Action": "s3:*",
"Resource" : [
"arn:aws:s3:::conf-*" ,
"arn:aws:s3:::conf-*/*"
]
}
]
}
```

Permission Boundaries:

A permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity. An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

You can use an AWS managed policy or a customer managed policy to set the boundary for an IAM entity (user or role). That policy limits the maximum permissions for the user or role.

For example, assume that the IAM user named `ShirleyRodriguez` should be allowed to manage only Amazon S3, Amazon CloudWatch, and Amazon EC2. To enforce this rule, you can use the following policy to set the permissions boundary for the `ShirleyRodriguez` user:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*",
        "cloudwatch:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```

When you use a policy to set the permissions boundary for a user, it limits the user's permissions but does not provide permissions on its own. In this example, the policy sets the maximum permissions of ShirleyRodriguez as all operations in Amazon S3, CloudWatch, and Amazon EC2. Shirley can never perform operations in any other service, including IAM, even if she has a permissions policy that allows it. For example, you can add the following policy to the ShirleyRodriguez user:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

IAM ARN's (Amazon Resource Name)

Please refer to the PPT shared for the ARN syntax. Few examples of ARN include:

arn:aws:iam::123456789012:root arn:aws:iam::123456789012:user/Bob
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Bob
arn:aws:iam::123456789012:group/Developers
arn:aws:iam::123456789012:group/division_abc/subdivision_xyz/product_A/Developers

arn:aws:iam::123456789012:role/S3Access
arn:aws:iam::123456789012:policy/ManageCredentialsPermissions
arn:aws:iam::123456789012:instance-profile/Webserver arn:aws:sts::123456789012:federated-user/Bob arn:aws:sts::123456789012:assumed-role/Accounting-Role/Mary
arn:aws:iam::123456789012:mfa/BobJonesMFA arn:aws:iam::123456789012:server-certificate/ProdServerCert arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/Prod ServerCert

Limitation of IAM Entities:

Policy documents can contain only the following Unicode characters: horizontal tab (x09), linefeed (x0A), carriage return (x0D), and characters in the range x20 to xFF.

Names of users, groups, roles, policies, instance profiles, and server certificates must be alphanumeric, including the following common characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-).

Path names must begin and end with a forward slash (/)

Policy names for inline policies (p. 179) must be unique to the user, group, or role they are embedded in, and can contain any Basic Latin (ASCII) characters, minus the following reserved characters: backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space.

User passwords (login profiles) can contain any Basic Latin (ASCII) characters

AWS account ID aliases must be unique across AWS products, and must be alphanumeric following DNS naming conventions. An alias must be lowercase, it must not start or end with a hyphen, it cannot contain two consecutive hyphens, and it cannot be a 12-digit number.

The following are the default maximums for your entities:

• Groups per AWS account: 100

• Users per AWS account: 5000 If you need to add a large number of users, consider using temporary security credentials.

• Roles per AWS account: 250

• Instance profiles per AWS account: 100

• Roles per instance profiles: 1 (each instance profile can contain only 1 role)

• Number of groups per user: 10 (that is, the user can be in this many groups)

• Access keys per user: 2

• Signing certificates per user: 2

• MFA devices in use per user: 1

• MFA devices in use per AWS account (at the root account level): 1

- Virtual MFA devices (assigned or unassigned) per AWS account: equal to the user quota for the account
- Server certificates per AWS account: 20
- AWS account aliases per AWS account: 1
- Login profiles per user: 1
- SAML providers per account: 100
- Identity providers (IdPs) per SAML provider: 10
- Keys per SAML provider: 10
- Customer managed policies per AWS account: 1000
- Versions per managed policy: 5
- Managed policies attached per IAM user, group, or role: 2

Maximum length for entities:

- Path: 512 characters
- User name: 64 characters
- Group name: 128 characters
- Role name: 64 characters
- Instance profile name: 128 characters
- Unique ID (applicable to users, groups, roles, managed policies, and server certificates): 32 characters
- Policy name: 128 characters
- Certificate ID: 128 characters
- Login profile password: 1 to 128 characters
- AWS account ID alias: 3 to 63 characters.
- Role trust policy (the policy that determines who is allowed to assume the role): 2,048 characters

For inline policies: You can add as many inline policies as you want to a user, role, or group, but the total aggregate policy size (the sum size of all inline policies) per entity cannot exceed the following limits:

- User policy size cannot exceed 2,048 characters
- Role policy size cannot exceed 10,240 characters
- Group policy size cannot exceed 5,120 characters Note IAM does not count whitespace when calculating the size of a policy against these limitations

For Managed Policies:

You can add up to two managed policies to a user, role, or group. The size of each managed policy cannot exceed 5,120 characters.

Note: Cross check above values from the AWS official page as this is the count when document is prepared.

You can request to increase some of these quotas for your AWS account on the ***IAM Limit Increase Contact Us Form***.

IAM does not count whitespace when calculating the size of policy against this limitation.

Let's see few Example of different Policies in AWS:

Example: Policy which allows a user to manage his own access key

{

“Version”: “2020-02-12”,

"Statement": [{

"Effect": "Allow",

"Action": ["iam:*AccessKey*"],

"Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/division_abc/sub division_xyz/sachin" }]

}

Example: IAM: Allows and Denies Access to Multiple Services Programmatically and in the Console

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowServices",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam:*LoginProfile*"
            ],
            "Resource": ["arn:aws:iam::*:user/sachin"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2:*",
            "Resource": "arn:aws:ec2:*:*:instance/i-1234567890abcdef0"
        }
    ]

}
```

In the above policy, we see that it allows full access to several services and limited self- managing access in IAM. It also denies access to the Amazon S3 logs bucket or the Amazon EC2 i-1234567890abcdef0 instance.

Following are the definition of the statements used in above policy:

- **AllowServices** : It allows full access to the specified AWS services. This means that user's actions in these services are limited only by permission policies that are attached to the user.
- **AllowIAMConsoleForCredentials** : It allows access to list all IAM users. This access is necessary to navigate the Users page in the AWS Management Console. It also allows viewing the password requirements for the account, which is necessary for the user to change their own password.
- **AllowManageOwnPasswordandAccessKeys** : It allows users manage only their own console password and programmatic access keys. This is important because if another policy gives a user full IAM access, that user could then change their own or other users' permissions. This statement prevents that from happening.
- **DenyS3Logs** : This explicitly denies access to the logs bucket. This policy enforces company restrictions on the user.
- **DenyEC2Production** : This explicitly denies access to i-1234567890abcdef0 instance.

This policy does not allow access to other services or actions. When the policy is used as a permissions boundary on a user, even if other policies attached to the user allow those actions, AWS denies the request.

Example 2:

S3: Allows Read and Write Access to Objects in an S3 bucket.

In this example, we see to create a policy that allows read and write access to objects in a specific S3 bucket. This policy grants the permissions necessary to complete this action from AWS API or AWS CLI only.

The s3:* object action is basically part of action name ending with object. The **AllowObjectActions** statement allows the GetObject, DeleteObject, PutObject and any other amazon S3 actions that ends with the word "Object".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": ["arn:aws:s3:::bucket-name"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource": ["arn:aws:s3:::bucket-name/*"]
        }
    ]

}
```

```
To allow read and write access from console, please add below statements:

"Statement": [
        {
            "Sid": "ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicyStatus",
                "s3:GetBucketPublicAccessBlock",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },

     For more examples on policies visit:
```

https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_examples.html

Exam Tips:

- IAM is Universal. It does not apply to regions at this time.
- A Root account is simply the account created when first setting up an AWS account. It has complete admin access and **should not** be used for day-to-day activities.
- When first created, new users have *NO permissions. Permissions must be explicitly given via group or policy.
- New users are assigned Access Key ID & Secret Access Key created by admin when user is created. These are not the same as Password and login id which can be used to login to the AWS console, instead they are used to access AWS via command-line or the API.
- You need to regenerate the Access Key ID and Secret Access key if you lose them.
- **Always** setup MFA on your root account.

- You can configure your own password policies.
- Remember that there are three different types of IAM Policies:
- Managed Policy: AWS-managed default policies
- Customer Managed Policy: Managed by you
- Inline Policy: Managed by you and embedded in a single user, group, or role.
- In most cases, AWS recommends using Managed Policies over Inline Policies.

For any issue, visit here to look at most frequently issues:

https://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot.html