INTRODUCTION

AWS cloud services provides secure virtual platform for users to deploy their applications. This service has helped organizations as well as individuals to provide robust and highly available services to their end-users. While conventional on-premise application deployments guarantee a certain level of security for the users, AWS security provides a high level data protection at a fraction of the cost to its users. There are quite a handful of security services on offer to AWS users, but Identity and Access Management (IAM) is one of the most widely used.

AWS IAM enables you to securely control access to AWS services and resources for your users. By implementing IAM, you can create Users and groups and assign permissions to allow or deny their access to AWS resources.

How IAM Works

1. A principal is an entity that can perform an action on AWS resources. The principal can be a user, a role, or an application
2. Authentication is the process of confirming the Identity of the principal that is trying to access an AWS resource. The principal must provide a credential or keys for authentication
3. A principal then sends a request to AWS, stating the actions it intends to perform and which resource is to perform it
4. By default all actions on all resources are denied. IAM authorizes a request only if all parts of the request are allowed by a matching policy. After authenticating and authorizing the request, AWS approves the action.
5. Actions are used to view, create, edit or delete a resource
6. Resources: a set of actions can be performed on a resource related to your AWS account.

Components of IAM

There are three main components of IAM namely: the User or group of users, Policies and Roles.

Users: An IAM identity with associated credentials and permissions attached to it. A user could be an actual person or an application.

Groups: A collection of IAM users make up an IAM group. Creating a group of users makes user permission management easier. By assigning permission to a group, all users in the group are assigned the said permissions, instead of managing permissions individually.

Policies: IAM policies sets permission and control access to AWS resources. Policies are stored in JSON formats. Permissions specifies who has access to a resource and what actions they can perform. There are 2 types of policies, Managed and Inline policies. Managed policy is a default policy that you attach to multiple entities (users, groups, and roles) in your AWS account. Inline policies are policies that you create that are embedded directly into a single entity (user, group or role).

#Grants user Jane access point Get and Put permissions

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": {

"AWS": "arn:aws:iam::123456789012:user/Jane"

},

"Action": ["s3:GetObject", "s3:PutObject"],

"Resource": "arn:aws:s3:us-west-2:123456789012:accesspoint/my-access-point/object/Jane/\*"

}]

}

Roles: IAM role is a set of permissions that define what actions are allowed and denied by an entity in the AWS console. It is similar to a user in that it can be accessed by any type of entity (an individual or AWS service), but instead of being uniquely associated with a person, a role is expected to be assumed by anyone who needs it. It can be used for temporary access to AWS resources within an account.

Lab: IAM with Terraform

In the following lab session, we will be creating an admin user in our AWS account. For more information on usong Terraform to create Users and Policies, check [here](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_policy) and here

Create a file admin\_user\_group.tf in an empty directory. First we will create our administrator group, this will make it convenient to add users to this administrator group:

resource "aws\_iam\_group" "administrators" {

name = "Administrators"

path = "/"

}

Then we create a group policy attachment to be attached to our created group:

data "aws\_iam\_policy" "administrator\_access" {

name = "AdministratorAccess"

}

resource "aws\_iam\_group\_policy\_attachment" "administrators" {

group = aws\_iam\_group.administrators.name

policy\_arn = data.aws\_iam\_policy.administrator\_access.arn

}

Next we create an IAM user named Administrator

resource "aws\_iam\_user" "administrator" {

name = "Administrator"

}

Then we need to add our user to the IAM group we created ealier

resource "aws\_iam\_user\_group\_membership" "devstream" {

user = aws\_iam\_user.administrator.name

groups = [aws\_iam\_group.administrators.name]

}

We can also enable console login for the user and output the password:

resource "aws\_iam\_user\_login\_profile" "administrator" {

user = aws\_iam\_user.administrator.name

password\_reset\_required = true

}

output "password" {

value = aws\_iam\_user\_login\_profile.administrator.password

sensitive = true

}

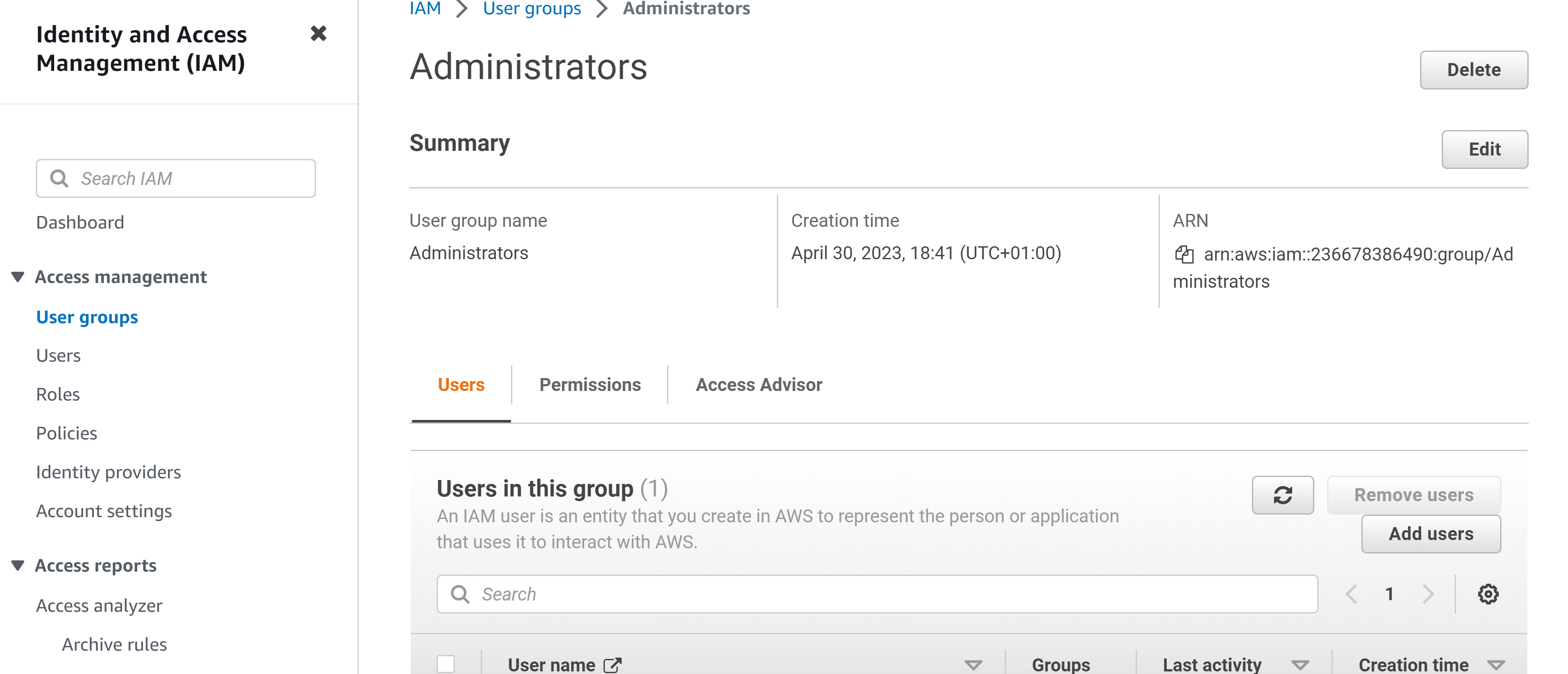
The next step is to initialize the working directory and apply our changes by executing these commands in the same directory as our Terraform file

>terraform init

>terraform apply

>terraform output password

When we check the console, we can confirm that the Administrator group was successfully created:



As well as the user we just created

